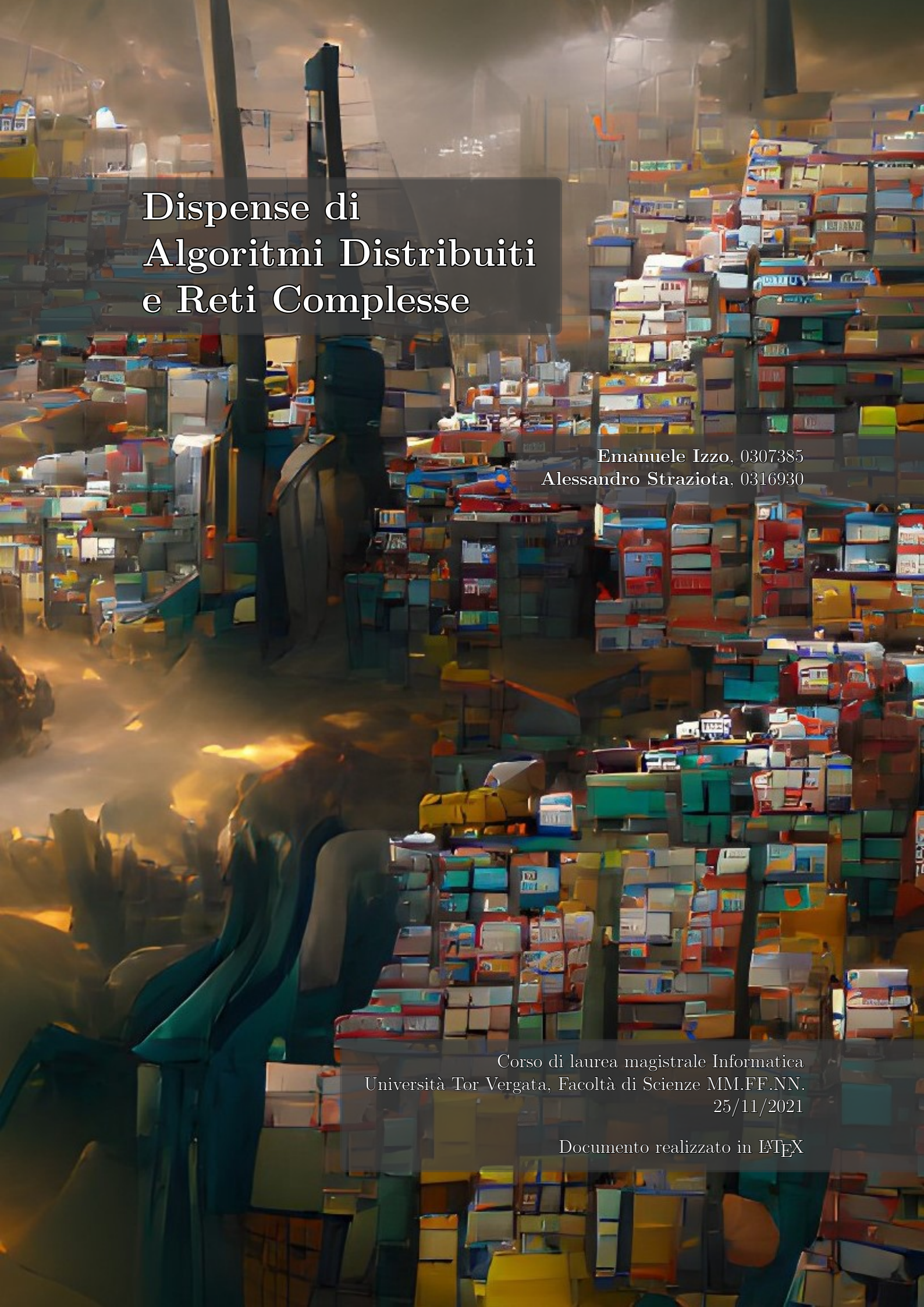


Dispense di Algoritmi Distribuiti e Reti Complesse

Emanuele Izzo, 0307385
Alessandro Straziota, 0316930

Corso di laurea magistrale Informatica
Università Tor Vergata, Facoltà di Scienze MM.FF.NN.
25/11/2021

Documento realizzato in LaTeX

Indice

Parte I

Protocolli deterministici distribuiti

Capitolo 1

Introduzione

1.1 Gli ambienti distribuiti

Informalmente è possibile definire come **ambiente distribuito** un *insieme di agenti*, aventi ognuno delle *capacità computazionali*, che interagiscono tra di loro scambiandosi messaggi tramite uan serie di *interconnessioni*. Talem ambiente è rappresentabile tramite un grafo in cui i nodi rappresentano gli agenti mentre gli archi rappresentano le connessioni tra di essi.

1.2 Le entità

Le **entità** sono gli agenti che svolgono azioni all'interno della rete; ogni entità possiede capacità computazionali, le quali sono definite come un insieme di operazioni possibili, tra cui:

- *Local storage*, ossia la capacità di conservare informazioni.
- *Processing*, ossie poter processare dati.
- *Scambio di messsaggi* con altri nodi della rete.
- *(Re)impostazione del clock*, il quale va a scandire i tempi in cui vengono eseguite le varie azioni.
- *Modifica dei registri*, i quali sono usati per eseguire operazioni o memorizzare lo stato attuale dell'entità.

1.2.1 Lo stato

Ogni enttià possiede uno **stato** interno, il quale rappresenta un'informazione necessaria per il corretto funzionamento di un protocollo distribuito. Più informalmente ogni nodo ha un insieme finito di stati possibili Σ definito a priori, e in ogni istante le entità si trovano in uno dei possibili stati (il quale dovrà essere sempre definito).

1.3 Gli eventi

Il comportamente di ogni entità è *reattivo*, ovvero si scatena in seguito all'avvenire di **eventi**. Alcuni esempi di eventi possono essere:

- Il tick del clock (il quale è un evento interno al nodo).
- La ricezione di un messaggio (il quale è un evento esterno al nodo).
- Un impulso esterno alla rete.

Anche se in un contesto reale gli eventi avvengono su una linea temporale discreta, essi verranno considerati essere avvenuti in maniera *sequenziale* (o ordinata). Questo perché, in un intervallo di tempo continuo, la probabilità che due eventi indipendenti occorrano nello stesso identico istante è pari a 0.

1.4 Le azioni

Un'**azione** è una sequenza di task (o attività) da svolgere in base allo stato attuale in cui l'entità si trova e all'evento a cui l'agente reagisce. Le task che un'agente può eseguire sono per esempio:

- Computare un algoritmo.
- Mandare un messaggio.
- Cambiare stato.

Un'azione è considerata come *atomica*, ovvero una volta iniziata non può essere interrotta in alcuna maniera. Inoltre è importante specificare che ne lostro modello le azioni *terminano sempre in un tempo finito.*

1.5 Il comportamento delle entità

Con **comportamento** di un'entità si intende un insieme di regole definite dalla funzione $B(x) : \Sigma \times \mathcal{E} \mapsto A$, dove Σ è l'insieme degli stati possibili, $\mathcal{E}$ è l'insieme degli eventi ed A è l'insieme delle azioni che può eseguire. Un dettaglio importante della funzione $B(x)$ è che è *deterministica* e *completa*, ossia $\forall (s, e) \in \Sigma \times \mathcal{E}, \exists ! \, a \in A$

1.6 Il protocollo

Raggruppando tutti i comportamenti delle entitá di un sistema è possibile definire il cosidetto **protocollo distribuito** (o **algoritmo distribuito**). Più formalmente, un protocollo è definito dall'insieme $B =: \{B(x) : x \text{ è un'entità del sistema}\}$. Un sistema si dice **simmetrico** (o **omogeneo**) se ogni entità ha uno stesso comportamento, ossia $\forall x, y$ entità, $B(x) \equiv B(y)$. È da notare che è possibile trasformare un qualsiasi sistema in un sistema omogeneo.

Consideriamo un sistema dove esistono due tipi di entità: entità *server* ed entità *client*. Per ogni entità definiamo un *ruolo*, il quale definisce il tipo dell'entità, e in base a questo eseguiamo l'azione corrispondente. Di seguito è riportato un pezzo di codice Python che riproduce questo comportamento:

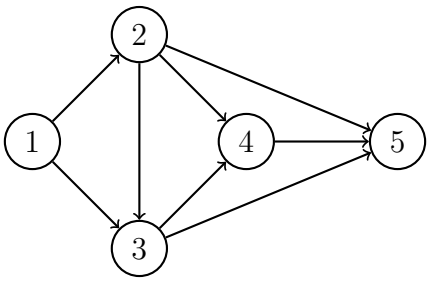
Esempio di protocollo omogeneo

```
class Entity:
    def __init__(self, role):
        self.role = role

        def action(self, state, event):
            if self.role == 'server':
                server_action(self, state, event)
            else:
                client_action(self, state, event)
```

1.7 Le comunicazioni

Le entità di una rete distribuita possono (e devono) interagire tra di loro. Il modo in cui interagiscono è tramite uno **scambio di messaggi** (o **message passing**), dove un messaggio è inteso essere una *sequenza finita di bit*. I messaggi viaggiano su una **rete di comunicazione**, ovvero un grafo *diretto* che specifica quali nodi possono scambiarsi messaggi.



È importante specificare che i nodi della rete non hanno una visione *globale* della rete, bensì hanno la sola visione **locale**. Più formalmente questo significa che ogni nodo può vedere solamente due gruppi di *porte logiche*:

- $N_{out}(x) \subset V$, ossia l'insieme dei vicini in uscita dell'entità x, dalle quali porte logiche usciranno i messaggi in uscita.
- $N_{in}(x) \subset V$, ossia l'insieme dei vicini in entrata dell'entità x, dalle quali porte logiche arriveranno i messaggi in entrata.

Secondo queste definizione il **vicinato** di un nodo x sarà definito dall'unione dei due insiemi, ossia $N(x) = N_{out}(x) \cup N_{in}(x)$.

1.8 Gli assiomi della comunicazione

È possibile identificare delle ipotesi fondamentali quando si discute dei protocolli:

- In assenza di **perdite di messaggi**, si assume che il tempo di trasmissione di un messaggio da un nodo verso un suo vicino sia *finito* ma *sconosciuto*. Per ogni messaggio `msg` esiste un tempo $t \in \mathbb{R}^+$ tale che entro il tempo t il messaggio `msg` viene trasmesso lungo un arco.

- Ogni nodo riesce a *distinguere* i suoi vicini, ovvero sa quale porta arriva un messaggio o attraverso quale porta inviarlo. In particolare, per ogni arco (x, y) esistono due etichette $\lambda_x(x, y)$ per il nodo x e $\lambda_y(x, y)$ per il nodo y. È molto importante notare che non è detto che $\lambda_x(x, y) = \lambda_y(x, y)$, ovvero l'etichettamento di un arco è solamente *locale* ad un nodo. Inoltre, non è nemmeno detto che $\lambda_x(x, y) = \lambda_x(y, x)$, ovvero un nodo non sa distinguere se una coppia di archi in entrata e uscita lo connettono ad uno stesso vicino o meno. Secondo questa definizione di orientamento locale, è possibile definire la **topologia** di un grafo come un grafo *etichettato* $(G, \lambda_v)_{v \in V}$.

1.9 Le restrizioni del modello

Spesso si definiscono protocolli secondo l'assunzioni di ulteriori restrizioni. Ovviamente più restrizioni ci sono più il protocollo è debole, in quanto togliend tali restrizioni il protocollo potrebbe non funzionare. Un protocollo con meno restrizioni risulta invece più generale e perciò più forte.

1.9.1 Restrizioni di comunicazione

- **Message ordering**: in assenza di perdita di messaggi, i messaggi trasmessi mediante uno stesso link arrivano nello stesso ordine, seguendo quindi una coda di tipo *FIFO* ordinata con l'ordine di invio (e non di ricezione).

- **Bidirectional links**: in questo modello le connessioni sono di tipo *bidirezionale*, e ogni nodo è anche in grado di distinguere gli estremi degli archi. Più formalmente

$$\forall x \in V, N_{out}(x) = N_{in}(x) = N(x) \wedge \forall y \in N(x), \lambda_x(x, y) = \lambda_x(y, x)$$

1.9.2 Restrizioni di affidabilità

- **Guaranteeed delivery**: i messaggi non arrivano mai corrotti.

- **Partial reliability**: non ci saranno *fallimenti* dal momento in cui osservo il sistema in poi (ciò non implica che prima di tale momento non ci siano stati fallimenti).

- **Total reliability**: ho la certezza che non sono mai avvenute failures e mai avverranno.

1.9.3 Restrizioni di topologia

- **Connection**: il grafo G è connesso.

1.9.4 Restrizioni di conoscenza

- **Knowledge of number of nodes**.
- **Knowledge of number of links**.
- **Knowledge of diameter**.

1.9.5 Restrizioni di tempo

- **Bounded communication delay**: rsiste una costante Δ tale che, in assenza di failures, il tempo tempo di trasmissione dei messaggi tramite qualsiasi connessione è al più Δ.
- **Synchronized clocks**: ogni clock delle entità è incrementato di una unità temporale costante in maniera simultanea.

1.10 Le misure di complessità

In un sistema distribuito non ha sempre senso parlare di *time complexity*, in quanto non è detto che il tempo di trasmissione di un messaggio sia sempre lo stesso, e soprattutto non è detto che il clock sia uguale per tutti. Per far sì che abbia senso parlare di time complexity si necessitano di due restrizioni: la *synchronized clocks restriction* e la *bounded communication delay restricion* (idealmente, tutti i messaggi dovrebbero viaggiare con uno stesso delay unitario). Un'altro tipo di complessità che verrà quasi sempre analizzata è la cosidetta **message complexity**, ovvero una complessità che tiene conto del numero di messaggi trasmessi prima che un protocollo termini un suo task. Notare che, se un messaggio per arrivare dal nodo a al nodo b necessita di essere instradato 4 volte, allora verranno contante tutte e 4 le trasmissioni fatte prima di arrivare a b, anche se il messaggio è uno solo.

Capitolo 2

Il problema del broadcast a singola sorgente

2.1 La definizione del problema

Dato un grafo non diretto $G := (V, E)$, un messaggio $m \in \{0,1\}^*$ e una funzione di stato $s : V \to \{\texttt{informed}, \texttt{not\_informed}\}$, si vuole progettare un protocollo che propaghi il messaggio a tutti i nodi. Una configurazione iniziale prevede la presenza di un solo nodo sorgente informato, mentre una configurazione finale prevede che tutti i nodi del grafo siano informati; per questo problema lavoriamo sotto le assunzioni che esiste un solo inizializzatore, il sistema è totalmente affidabile, i link sono bidirezionali e il grafo G è connesso. È necessario siepcificare i nodi passano dallo stato `not_informed` ad `informed` nel momento in cui ricevono un messaggio. Si può osservare che, mentre possono esistere n configurazioni inziali possibili, la configurazione finale è unica. Inoltre è sottintanteso che l'inizializzatore unico corrisponda al primissimo nodo informato (in realtà questo non è sempre vero, infatti potrebbe capitare un problema in cui l'inizializzatore non corrisponda col primo nodo informato).

2.2 Il protocollo di flooding

Una prima idea banale è la seguente: se un'entità riceve un messaggio allora lo invia ai suoi vicini. Intuitivamente quest'idea porterà ad una configurazione finale basandosi sul solo fato che il grafo è connesso. Possiamo perciò definire lo stato `INITIATOR`, nel quale si trova l'unico nodo iniziale col messaggio e che si occuperà di far iniziare il protocollo, e lo stato `SLEEPING`, in cui si trovano gli altri nodi in attesa di ricevere il messaggio

Esempio di codice per il protocollo `flood`

```
class Entity:
    def __init__(self, state, message):
        self.state = state
        self.message = message

    def action(self):
        if self.state == "INITIATOR":
            spontaneously:
                send(self.message) to N(self)
        elif self.state == "SLEEEPING":
            receiving(message):
                self.message = message
                send(message) to N(self)
```

2.2.1 Correttezza

Teorema (2.1): se eseguiamo il protocollo `flood` sotto le assunzioni fatte prima, esiste un tempo $t \in \mathbb{R}^+$ tale che ogni nodo di G ha ricevuto il messaggio m almeno una volta, ovver $\forall v \in V, s(v) = \texttt{informed}$.

Teorema (2.1): Sia L_d l'insieme di nodi distanti esattamente d dalla sorgente s. Si vuole dimostrare per induzione su d che esiste un tempo t_d tale che tutti i nodi in L_d hanno ricevuto almeno una volta il messaggio, ovvero per ogni nodo $v \in L_d$ al tempo t_d sia $s_{t_d}(v) = \texttt{informed}$. Per $d = 0$ è banale in quanto $L_0 := \{s\}$, perciò come base prendiamo $d = 1$. Per definizione avremo che $L_1 := N(s)$. Per come è descritto il protocollo, la sorgente s ad un certo punto invierà in maniera spontanea il messaggio a tutti i nodi in $N(s)$, ergo L_1. Dato che stiamo sotto l'assunzione di affidabilità totale, siamo certi che non ci saranno fallimenti durante la trasmissione, quindi per $d = 1$ certamente esisterà un istante $t_1 \in \mathbb{R}^+$ entro il quale tutti i nodi in L_1 verranno informati. Supponiamo ora che l'ipotesi sia vera fino ad un certo $d - 1$: si vuole dimostrare che sia vera anche per d, ovvero che $\forall v \in L_d, \exists t_d : s_{t_d}(v) := \texttt{informed}$. Consideriamo un qualsiasi nodo $v \in L_d$, allora per definizione esisterà un nodo $y \in L_{d-1}$ tale che $(y, x) \in E$ (sfruttando anche l'ipotesi di link bidirezionali). Se $y = s$ allora vuol dire che $x \in L_1$, riconducendosi quindi al caso base. Se invece $y \neq s$, allora certamente y avrà ricevuto il messaggio mentre era nello stato `SLEEPING`, e per definizione del protocollo y avrà inviato il messaggio al suo vicinato $N(y)$, compreso x. Sfruttando ancora l'ipotesi di affidabilità totale sappiamo che il messaggio arriverà in un tempo finito e intato a x tramite y. Infine, sfruttando l'ipotesi di connettività di G, sappiamo che $\forall v \in V, \exists k \in [0, diam(G)] : v \in L_k$.

2.2.2 Miglioramento del protocollo

Per come è stato descritto, il protocollo precedente porta a termine il suo task, senza però termina mai. Inoltre, quando un nodo instrada un messaggio ricevuto a tutto il suo vicinato, esso rispedisce una copia superflua anche al suo mittente (per via dei link bidirezionali). Per ovviare a questo bastano le seguenti modifiche:

- Se il nodo ha ricevuto almeno una volta il messaggio, allora passerà nello stato `DONE`, nel quale non ci sarà più bisogno che inoltri ulteriori messaggi (preché è già stato fatto in precedenza).
- Un nodo che riceve un messaggio non lo inoltra a chi glielo ha inviato. Sappiamo che un nodo può distinguere i nodi agli estremi dei suoi archi incidenti grazie all'assunzione di link bidirezionali, ovvero $\forall x \in V, \forall y \in N(x), \lambda_x(x, y) = \lambda_x(y, x)$.

Esempio di codice per il protocollo `optimized-flood`

```
class Entity:
    def __init__(self, state, message):
        self.state = state
        self.message = message

    def action(self):
        if self.state == "INITIATOR":
            spontaneously:
                send(self.message) to N(self)
                self.state = "DONE"
        elif self.state == "SLEEEPING":
            receiving(message):
                self.message = message
                send(message) to N(self)
                self.state = "DONE"
        elif self.state == "DONE":
            None
```

2.2.3 Message complexity

Di seguito sono proposte due analisi abbastanza semplici sul protocollo ottimizzato, le quali porteranno ad un risultato asintoticamente uguale, ma con un livello di dettaglio differente.

- **Analisi 1:** si osservi che per via del ritardo di trasmissione, su ogni arco possono essere trasmessi al più due messaggi durante l'intero protocollo. Sappiamo che un nodo non invierà mai un messaggio attraverso l'arco da cui ha ricevuto, però per via del ritardo di trasmissione potrebbe capitare di inviare un messaggio su un arco nel quale sta già viaggiando un altro messaggio in etrata. Fatta questa premessa, nel caso peggiore verranno inviati due messaggi per ogni arco, perciò il protocollo avrà complessità al più $2m \in O(m)$.

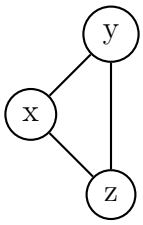
- **Analisi 2:** Nel caso precedente si è calcolata la complessità in base gli archi, volendo però è possibile calcolare la complessità in base ai nodi. Definiamo con $msg(v)$ il numero complessivo di messaggi inviati dal nodo v, e con $msg(\texttt{FLOOD})$ il numero di messaggi trasmessi durante il protocollo (ovvero la sua message complexity).

$$\begin{aligned} msg(\texttt{FLOOD}) &= \sum_{v \in V} msg(v) = \\ &= msg(s) + \sum_{v \in V - \{s\}} msg(v) = \\ &= |N(s)| + \sum_{v \in V - \{s\}} (|N(v)| - 1) = \\ &= |N(s)| + \sum_{v \in V - \{s\}} |N(v)| - \sum_{v \in V - \{s\}} 1 = \\ &= (\sum_{v \in V - \{s\}} |N(v)|) - (n-1) = \\ &= (\sum_{v \in V - \{s\}} deg(v)) - (n-1) = \\ &= 2m - n + 1 \end{aligned}$$

  Come si può vedere, da questa analisi emerge che il protocollo risparmia $n - 1$ messaggi trasmessi.

2.3 I tempi e gli eventi

Dato uno *stesso input* ad un *algoritmo deterministico centralizzato*, esso non genererà uno *stesso aoutput*, ma la *sequenza di azioni* che svolgerà per raggiungere tale output sarà sempre la stessa, anche se viene rieseguito infinite volte. La stessa cosa non si può sempre dire di un *algoritmo deterministico distribuito*. Infatti nel protocollo di floodingm dato che il delay dei messaggi varia in maniera indeterminata, avremo che le **traiettorie** (o **sequenze di azioni**) verso l'unica configurazione finale varia da esecuzione ad esecuzione, anche a fronte di una stessa configurazione iniziale. Consideriamo il seguente grafo:



Consideriamo il nodo x come nodo sorgente. Eseguendo per la prima volta il protocollo di flooding potrebbe capitare che i ritardi di trasmissione sugli archi (x, y) e (x, z) siano molto simili, perciò una possibile sequenza di azioni potrebbe essere la seguente:

Figura 2.1: Esempio di esecuzione dell'algoritmo di flooding.

Eseguendo il protocollo una seconda volta, potrebbe capitare che il ritardo di trasmissione sull'arco (x, y) sia così tanto alto che alla fine y riceve prima il messaggio tramite z anziché x.

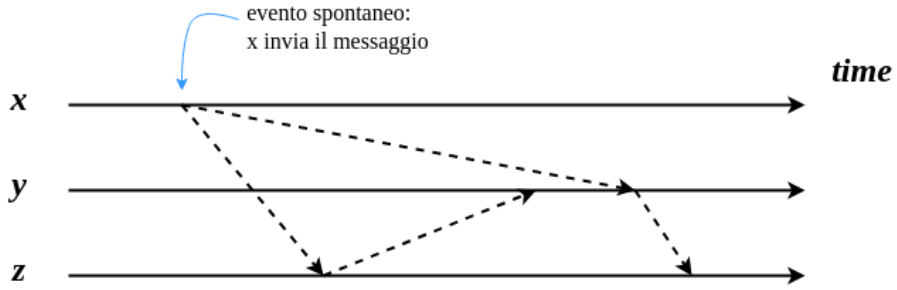


Figura 2.2: Esempio di esecuzione dell'algoritmo di flooding.

Perciò, le due precedenti esecuzioni possono essere rappresentate come dei **grafi di comunicazione**.

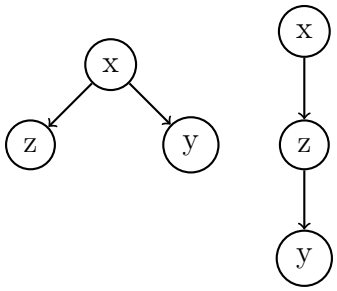


Figura 2.3: Due possibili grafi di comunicazione.

Dato che il grafo di comunicazione è sostanzialmente un *albero ricoprente*, possiamo avere quindi $O(n^n)$ possibili sequenze di trasmissioni (o traiettorie) che portano alla configurazione finale.

2.4 Un lower-bound alla message complexity

Definiamo con $\sigma(x, t)$ lo **stato locale interno** di un nod x al tempo t. Per stato interno si intende qualsiasi informazione contenuta all'interno di x, come il contenuto della memoria, il contenuto dei registri, il registro di stato, etc.

Osservazione: Se un nodo x si trova in due ambienti differenti E_1, E_2 ma il suo stato interno è identico, allora x non sarà in grado di distinguere i due ambienti ed eseguirà la stessa azione.

Osservazione: Se due nodi distinti x, y ad un certo punto raggiungono lo stesso stato $\sigma(x,t) = \sigma(y,t)$, allora se sollecitati da uno stesso evento eseguiranno le stesse azioni.

Teorema (2.2): sotto le ipotesi di unico inizializzatore, affidabilità totale, link bidirezionali e connettività, ogni protocollo deterministico di broadcasting richiede di scambiare almeno $m = |E|$ messaggi.

Teorema (2.2): Supponiamo per assurdo che esista un protocollo P con message complexity esattamente $msg(P,G) = m-1 < m$. Perciò, deve esistere un arco (x,y) attraverso il quale non viene trasmesso alcun messaggio. Certamente possiamo dire che nè x e nè y sono inizializzatori, in quanto se lo fossero certamente un messaggio sarebbe stato trasmesso sull'arco (x,y) (per l'ipotesi di affidabilità totale). A questo punto rimuoviamo l'arco (x,y) e aggiungiamo un nuovo nodo z insieme agli archi $(x,z),(y,z)$. Rieseguiamo il protocollo con gli stessi ritardi di trasmissione (coas necessaria per il discorso affrontato prima). A livello di stato locale interno, nè al nodo x nè al nodo y è cambiato nulla (*seconda osservazione*). Infatti, sia x che y vedranno un arco uscente laddove prima c'era l'arco (x,y). Però sappiamo che i nodi non sanno assolutamente distinguere i nodi agli estremi degli archi. Perciò, per quanto ne sa x, dall'altra parte del nuovo arco ci sta ancora y anziché z (e viceversa). Però, sapendo che da quella porta nè x nè y trasmettono, allora certamente nessuno dei due trasmetterà sui relativi archi che li collegano a z. Infatti i due ambienti, dal punto di vista locale, sono identici, perciò x ed y compieranno le stesse azioni (*prima osservazione*), ovvero non trasmettere nulla. Dato che z è collegato solamente ad x ed y, esso certamente non può ricevere il messaggio da un altro nodo, perciò z non verrà mai informato, e quindi il protocollo P non è corretto.

2.5 Il problema del broadcast a singola sorgente su ipercubi

Generalmente in un contesto reale si vuole costruire reti che abbiano le seguenti proprietà:

- un *diametro* piccolo ($O(1)$ o $O(\texttt{POLY}(\log n))$), in modo tale da ridurre i tempi di comunicazione tra i nodi.
- Il grafo deve essere *sparso*, risparmiando in termini di risorse per la creazione di link.

Da notare che, riducendo il grado massimo di un grafo, si ottiene una sua sparsificazione aumentandone anche la sua *scalabilità* (la capacità di non comportare eccessivi costi nel caso di aggiunta di un nuovo nodo). Esaminiamo alcuni grafi particolari:

- Una *clique* K_n, seppur ha un diametro basso ($d = 1$) e sia resistente (è necessario rompere almeno $n - 1$ links), non è molto scalabile in quanto ogni volta che viene aggiunto un nuovo nodo bisogna creare $n - 1$ links.

- Una *lista*, seppur sia un grafo scalabile e sparso, presenta un diametro alto ($d = n-1$) e risulta poco resistente (basta che uno solo dei link si rompa per disconnettere il grafo).

- Una *griglia bidimensionale* comosta da $\sqrt{n}$ righe e colonne risulta più equilibrata, poiché il grado massimo è $d = 4$ mentre il diametro è $2\sqrt{n}$; inoltre è abbastanza scalabile, potendo collegare differenti griglie affiancandole e aggiungendo $\sqrt{n}$ archi.

- Un *ipercubo d-dimensionale* offre un buon compromesso in termini di grado massimo, densità, resistenza, diametro è scalabilità; il grado massimo è $d = \log_2 n$, inoltre tutti i nodi possiedono grado esattamente d, e pertanto il numero di archi è $\frac{n \log_2 n}{2}$. Risulta anche facilmente scalabile, poiché l'unione di due ipercubi d-dimensionali richiede solo n archi per unire i nodi analoghi.

Grafo	**Diametro**	**Grado**
Clique	$\Theta(1)$	$\Theta(n)$
Lista	$\Theta(1)$	$\Theta(1)$
Griglia	$\Theta(\sqrt{n})$	$\Theta(1)$
Ipercubo	$\Theta(\log n)$	$\Theta(\log n)$

2.5.1 Ipercubi *d*-dimensionali etichettati

Un **ipercubo d-dimensionale etichettato** $H_d = (V, E)$ è definito nel seguente modo:

- Ogni nodo è univocamente etichettato con una stringa in $\{0, 1\}^d$. Formalmente, l'insieme dei nodi è definito come $V := \{\bar{x} | \bar{x} \in \{0, 1\}^d\}$.

- Due nodi sono connessi da un arco se e solo se la loro **distanza di Hamming** è pari ad 1, dove la distanza di Hamming è definita come il numero di bit per cui 2 stringhe differiscono ($d_H(\bar{x}, \bar{y}) = \sum_{i=1}^{d} |x_i - y_i|$). Formalmente, l'insieme degli archi è definito come $E := \{(\bar{x}, \bar{y}) \in V^2 | d_H(\bar{x}, \bar{y}) = 1\}$.

- Ogni arco è etichettato con un numero nell'intervallo $[d]$, in modo tale che l'etichetta rappresenta l'indice del bit per il quale i due nodi estremi differiscono (partendo a contare per esempio dal bit meno significativo).

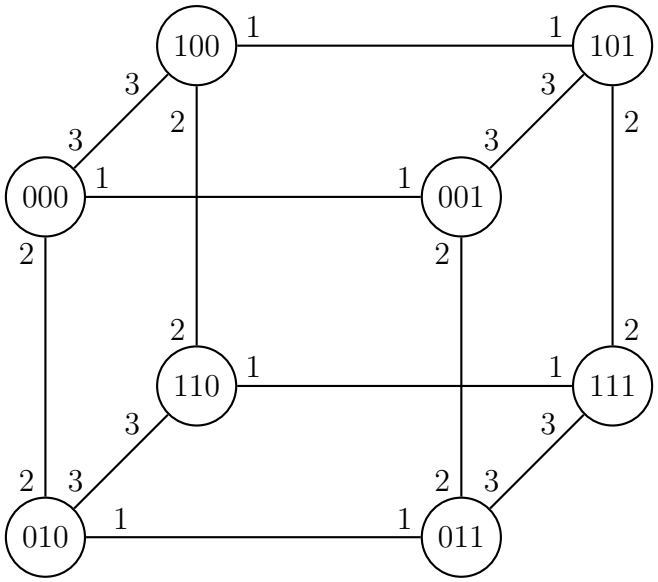


Figura 2.4: Un esempio di ipercubo 3-dimensionale etichettato.

Teorema (2.3): Un ipercubo d-dimensionale etichettato H_d ha diametro esattamente $d = \log_2 n$.

Teorema (2.3): Consideriamo due nodi differenti $\bar{x} =< x_1, x_2, ..., x_i, ..., x_d >$ e $\bar{y} =< y_1, y_2, ..., y_i, ..., y_d >$. Scorrendo in parallelo i caratteri delle due stringhe, prima o pio si arriverà ad un indice i_1 tale che $x_{i_1} \neq y_{i_1}$. Per costruzione dell'ipercubo, esiste un vicino di $\bar{x}$, chiamato $\bar{x}'$, tale che $x'_{i_1} = y_{i_1}$. Consideriamo quindi l'arco $(\bar{x}, \bar{x}')$ come primo arco del cammino P. A questo punto, $\forall i \geq i_1$ possiamo avere due casi:

- Le due stringhe $\bar{x}'$ e $\bar{y}$ corrispondono, e pertanto $\bar{x}' = \bar{y}$ e il percorso P è concluso.
- esiste un ulteriore indice $i_2 \geq i_1$ tale che $x'_{i_2} \neq y_{i_2}$. Per costruzione dell'ipercubo, esiste un vicino di $\bar{x}'$, chiamato $\bar{x}''$, tale che $x''_{i_2} = y_{i_2}$. Consideriamo quindi l'arco $(\bar{x}', \bar{x}'')$ come secondo arco del cammino P, e ripetiamo il procedimento.

Ripetendo questo passaggio t volte verrà raggiunto un nodo $\bar{x}^* = \bar{y}$ che conclude il percorso P. Nel caso peggiore, le due stringhe $\bar{x}$ e $\bar{y}$ differenziano per tutti i bit che le compongono: saranno pertanto necessari $t = d$ passaggi per arrivare a $\bar{y}$ da $\bar{x}$ nel caso peggiore. Questo ragionamento può essere anche fatto leggendo le stringhe da sinistra verso destra piuttosto che da destra verso sinistra.

2.5.2 Protocollo hyperflood

Il protocollo `hyperflood` è molto semplice e sfrutta fortemente il fatto che i nodi posseggono delle conoscenze globali della struttura della rete (in questa caso il fatto che si trovino in un ipercubo d-dimensionale etichettato). In maniera informale il protocollo funzione nella seguenta maniera:

- L'unico nodo `INITIATOR` invia il messaggio a tutti i suoi nodi vicini.
- Se un qualsiasi nodo riceve il messaggio da un arco etichettato con il numero l, allora inoltrerà il messaggio a tutti i suoi vicini connessi dagli archi con indice $l' > l$ (o con indice $l' < l$).

Esempio di codice per il protocollo `hyperflood`

```
class Entity:
    def __init__(self, state, message):
        self.state = state
        self.message = message

    def action(self):
        if self.state == "INITIATOR":
            spontaneously:
                send(self.message) to N(self)
        elif self.state == "SLEEEPING":
            receiving(message):
                self.message = message
                l = message.fromEdge
                for i in range(l+1, d):
                    send(message) to N(self)[i]
```

Questo protocollo può essere osservato in maniera ricorsiva. ovvero: quando un nodo riceve il messaggio dall'arco l diventa l'`INITIATOR` del sotto-ipercubo di dimensione $l' = d - l$ (o $l' = l - 1$).

2.5.3 Correttezza

Questo protocollo è corretto poiché esiste sempre un cammino con *etichette crescenti* da $\bar{x}$ ad $\bar{y}$. Nel protocollo `hyperflood` il messaggio verrà inoltrato esattamente attraverso quel cammino. Poiché si può invertire l'ordine di lettura delle stringhe, lo stesso ragionamento è possibile con un cammino di *etichette decrescenti*.

2.5.4 Message complexity

Teorema (2.4): La message complexity del protocollo `hyperflood` è ottima, ovvero vengono trasmessi esattamente $n - 1$ messaggi, uno per ogni nodo (sorgente esclusa).

Teorema (2.4): Supponiamo per assurdo che esiste un nodo $\bar{y}$ che riceve due copie del messaggio. Se così fosse, nel grafo delle comunicazioni esisterebbero due cammini

disgiunti che da un nodo $\bar{x}$ portando ad $\bar{y}$ (notare che $\bar{x}$ potrebbe anche essere la sorgente $\bar{s}$). Per costruzione dell'ipercubo etichettato, i due cammini iniziano per degli archi etichettati con l_1 ed l_2 tali che $l_1 < l_2$ (senza perdità di generalità poniamo che $l_1 < l_2$). Sia $\bar{x}'$ il nodo raggiunto da l_1 e $\bar{x}''$ il nodo raggiunto da l_2. Per costruzione avremo che:

- $\bar{x}$ differisce da $\bar{y}$ almeno per i bits in posizione l_1 ed l_2.
- $x'_{l_1} = y_{l_1}$ e $x'_{l_2} \neq y_{l_w}$.
- $x''_{l_1} = y_{l_1}$ e $x''_{l_2} \neq y_{l_w}$.

È importante osservare che entrambi i cammini procedono per etichette crescenti. Ciò significa che dal cammino passante per l_2 verranno toccati solo nodi che differiscono da $\bar{x}''$ per i bits con indice $l_i > l_2$. Questo implica che il cammino passante per l_2 porta al nodo $\bar{y}$ senza mai cambiare il bit in posizione l_1, e questo è un assurdo.

2.5.5 Time complexity

Teorema (2.5): In un ipercubo d-dimensionale etichettato sincrono nel quale il tempo di trasmissione vale un'unità temporale, la time complexity è ottima, ovvero $O(\log n)$.

Teorema (2.5): Dimostriamo per induzione suall distanza di Hamming dei nodi dalla sorgente: $\forall i \in [d], \forall x \in \{0,1\}^d : d_H(\bar{s}, \bar{x}) = i \Rightarrow x$ sarà informato al tempo i.

- Per $j - 1$ questo è vero perché per definizione del protocollo la sorgente $\bar{s}$ manderà il messaggio a tutti i suoi vicini, i quali formano l'insieme di tutti i nodi a distanza di Hamming 1 da $\bar{s}$.
- Per ipotesi induttiva supponiamo questo sia vero per $i - 1$, ovvero che tutti i nodi in L_{i-1} sono stati informati al tempo $t = i - 1$. Sappiamo che per ogni nodo $\bar{x} \in L_i$ esiste sempre un cammino con tutte le etichette crescenti dalla sorgente $\bar{s}$ ad $\bar{x}$. Dato però che le etichette sono crescenti, allora certamente $\bar{x}$ è raggiunto da un nodo $\bar{y}$ nell'insieme L_{i-1}. Per ipotesi induttiva $\bar{y}$ è stato informato al momento $i - 1$, inoltre il messaggio impiega un solo istante per percorrere l'arco $(\bar{y}, \bar{x})$, perció $\bar{x}$ sarà informato al tempo $t = (i - 1) + 1 = i$.

Capitolo 3

Il wake-up problem

3.1 La definizione del problema

Dato un grafo non diretto $G := (V, E)$, e una funzione di stato $s : V \to \{\texttt{awake}, \texttt{asleep}\}$, si vuole progettare un protocollo che porti tutti i nodi dallo stato `asleep` allo stato `awake`. Una configurazione iniziale prevede la presenza di un sottoinsieme di nodi $W \subset V$ nello stato `awake` e un sottoinsieme di nodi $\bar{W} = V - W$ nello stato `asleep`, mentre una configurazione finale prevede che tutti i nodi del grafo siano nello stato `asleep` (ossia $\bar{W} = V \cap W = \emptyset$); per questo problema lavoriamo sotto le assunzioni che il sistema è totalmente affidabile, i link sono bidirezionali e il grafo G è connesso. Se si pensa bene, il wake-up problem è una generalizzazione del problema del broadcasting, in cui non c'è un solo nodo inizializzatore, bensì un sottoinsieme di nodi. Più precisamente il problema del broadcasting è un caso particolare del wake-up problem in cui il sottoinsieme iniziale di nodi nello stato `awake` è composto da un solo nodo.

3.2 Il protocollo di wake-up

Come detto in precedenza, il problema di wake-up è una generalizzazione del problema del broadcast, perció applicando le dovute modifche al protocollo `flood` possiamo definire il protocollo `wake-up` per il problema.

Esempio di codice per il protocollo `wake-up`

```
class Entity:
    def __init__(self, state, message, is_initiator):
        self.state = state
        self.message = message
        self.is_initiator = is_initiator

    def action(self):
        if self.state == "AWAKE":
            if self.is_initiator:
                send("Wake-up!") to N(self)
            else:
                recieving(message):
                    None
        elif self.state == "ASLEEP":
            receiving(message):
                self.message = message
                send(message) to N(self)-message.sender
```

3.2.1 Correttezza

Teorema (3.1): Esuste un tempo $t \in \mathbb{R}^+$ tale che per ogni $x \in V$, lo stato di x al tempo t è $state_t(x) =$ `awake`, ovvero che al tempo t avremo che $W = K$.

Teorema (3.1): Sia $W \subset V$, definiamo la distanza di un nodo dall'insieme W come $d(x, W) = \min\{d(x, y) : y \in W\}$. Osserviamo che se $x \in W$ allora $d(x, W) = d(x, x) = 0$ Per definizione del protocollo sappiamo che ad un certo punto tutti i nodi in W che sono nello stato `awake` avranno inviato il messaggio a tutti i loro vicini, e data l'assunzione di total reliability sappiamo che esiste un tempo t_1 entro il quale tutti i noda a distanza 1 da W saranno informati (e di conseguenza svegliati). Per comodità definiamo l'insieme $L_1 := \{x \in V | d(x, W) = 1\}$. Per ipotesi induttiva supponiamo che per ogni $d > 1$ sia vero che esiste un tempo t_d tale che tutti i nodi in L_d siano nello stato `awake`. Si vuole dimostrare che questo è vor anche per il tempo $d + 1$. Consideriamo un nodo generico $x \in L_{d+1}$: certamente deve esistere almeno un arco (y, x) tale che $y \in L_d$. Se così non fosse, allora ogni cammino minimo di W arriverebbe in x tramite un arco (z, x) tale che $z \in L_k$ con $k \neq d$, e ciò implica che x è a distanza $k + 1 \neq d + 1$, ovvero $x \neq L_{d+1}$. Dato che y è stato informato entro il tempo t_d, e data la definizione del processo, y invierà il messaggio "Wake-up!" ad x. Infine, grazie alla totale affidabilità, siamo certi che tale messaggio arriverà a destinazione, svegliando x. Quindi esiste un tempo t_{d+1} tale che ogni $x \in L_{d+1}$ verrà svegliato entri il tempo t_{d+1}.

3.2.2 Message complexity

È possibile fare alcune osservazioni sul protocollo:

Osservazione: Nel caso peggiore in cui $W = V$ la messagge complexity sarà $2|E| = 2m$.

Osservazione: Nel caso migliore in cui $|W| = 1$, dove il protocollo di flooding è a singola sorgente, la message complexity sarà $2m - n + 1$.

Da qui abbiamo che $msg(\texttt{FLOOD}) = 2m - n + 1 \leq msg(\texttt{WAKE-UP}) \leq 2m$. Andando a fare un'analisi più fine, consideriamo il caso generico in cui $|W| = k$. Sicuramente k nodi invieranno i messaggi a tutti i loro vicini, mentre $n - k$ invieranno i messaggi a tutti i vicini eccetto al nodo dal quale sono stati svegliati.

$$\begin{aligned} msg(\texttt{WAKE-UP}) &= \sum_{w \in W} \delta(w) + \sum_{v \in V - W} (\delta(v) - 1) = \\ &= \sum_{w \in W} \delta(w) + \sum_{v \in V - W} \delta(v) - \sum_{v \in V - W} 1 = \\ &= \sum_{v \in V} \delta(v) - \sum_{v \in V - W} 1 = \\ &= 2m - (n - k) = 2m - n + k \end{aligned}$$

Dove $\delta(v)$ è il grado (uscente) del nodo v. Un caso particolare è quando il protocollo `wake-up` viene applicato un un albero. Sappiamo che il numero di archi di un albero è $m = n - 1$, pertanto la message complexity sarà $msg(\texttt{WAKE-UP}) = 2m - n + k = 2(n-1) - n + k = n + k - 2$.

3.3 Un lower-bound alla message complexity

Teorema (3.2): Se stiamo sotto le ulteriori assunzioni che G sia una clique K_n e che ogni nodo è univocamente identificabile dagli altri nodi da un identificatore globale che chiunque conosce, allora una delimitazione inferiore alla message compexity è $\Omega(n \log n)$.

Capitolo 4

Lo spanning tree construction

4.1 La definizione del problema

Uno **spanning tree** (o **albero ricoprente**) di un grafo $G := (V, E)$ è un sottoinsieme di archi $T \subset E$ tale che il grafo $G' := (V, T)$ risulta essere connesso e aciclico, ossia un albero. il problema che ci poniamo è di calcolare uno spanning tree in una reta in maniera distribuita. Le assunzioni sotto cui lavoriamo è che esista un solo inizializzatore, i link sono bidirezionali, la rete è totalmente affidabile, e il grafo G è connesso. Per ogni nodo $x \in V$ conserva un sottoinsieme di vicini denominato con `tree-neighbors`(x), che rappresenta i relativi vicini nello spanning tree. In una configurazione iniziale, per ogni nodo $x \in V$, l'insieme `tree-neighbors`(x) è vuoto; in una configurazione finale, per ogni nodo $x \in V$, `tree-neighbors`(x) contiene tutti i soli nodi vicini nello spanning tree ricavato dal protocollo.

4.2 Single source shout protocol

Il protocollo `single-source-shout` si basa sul far chiedere ad ogni nodo x se i suoi vicini appartengono già allo spanning tree: se il nodo già apparttiene allo spanning tree allora lo si scarta, se invece ancora non appartiene allora gli si propone se vuole diventare un nodo figlio. Un nodo, dato che può ricevere più richieste, risponderà `"YES"` alla prima risposta e `"NO"` alle altre. Definiamo quindi l'insieme degli stati del processo:

- $S := \{$`INITIATOR`, `IDLE`, `ACTIVE`, `DONE`$\}$.
- $S_{init} := \{$`INITIATOR`, `IDLE`$\}$.
- $S_{final} := \{$`DONE`$\}$.

Esempio di codice per il protocollo `single-source-shout`

```
class Entity:
    def __init__(self, state, message):
        self.state = state
        self.message = message

    def action(self):
        if self.state == "INITIATOR":
            spontaneously:
                self.root = True
                self.tree_neighbors = {}
                send(Q) to self.neighbors
                self.counter = 0
                self.state = "ACTIVE"
        if self.state == "IDLE":
            receiving(Q):
                self.root = False
                self.parent = Q.sender
                self.tre_neighbors = {Q.sender}
                send("YES") tp Q.sender
                self.counter = 1
                if self.counter == len(self.neighbors):
                    self.state = "DONE"
                else:
                    send(Q) to self.neighbors - {Q.sender}
                    self.state = "ACTIVE"
        if self.state == "ACTIVE"
            receiving(Q):
                send("No") to Q.sender
            receiving(R):
                if R.message == "YES":
                    self.tree_neighbors = self.tree_neighbors + {R.
    sender}
                    self.counter += 1
                elif R.message == "NO"
                    self.counter += 1
                if self.counter == len(self.neighbors):
                    self.state == "DONE"
        if self.state == "DONE":
            None
```

4.2.1 Terminazione del protocollo

Il protocollo `single-source-shout` è molto simile al protocollo `flood`, con l'aggiunta di messaggi feedback `"YES"` e `"NO"`. Dato che siamo sotto l'assunzione che G è connesso, allora certamente ogni nodo x (eccetto la sorgente) riceverà almeno una richiesta `Q` di

diventare figlio, perció ogni nodo nello stato `IDLE` passerà allo stato `ACTIVE`, con il proprio contatore pari a 1. Per ogni richiesta ricevuta tutti i nodi risponderanno `"YES"` o `"NO"`. Dato che prima di entrare nello stato `DONE` un nodo deve ricevere una risposta da tutti i vicini, e dato che per costruzione tutti rispondono alle richieste, allora prima o poi tutti i nodi passerranno dallo stato `ACTIVE` allo stato `DONE`, terminando localmente il proprio task.

4.2.2 Correttezza

Si vuole dimostrare che l'unione di tutti i `tree-neighbors`(x) è uno spanning tree per la rete G. Per prima cosa bisogna dimostrare la *coerenza* degli insiemi `tree_neighbors`(x), ovvero che se x ha come vicino y nello spanning tree ($y \in$ `tree_neighbors`(x)) allora anche allora nache x deve risultare essere vicino di y ($x \in$ `tree-neighbors`(y)). Questo è facilmente verificabile: certamente uno tra x e y deve necessariamente essere il nodo padre. Supponiamo che x sia il nodo padre senza perdita di generalità, dato che $y \in$ `tree-neighbors`(x), e dato che stiamo assumendo che y è figlio di x, allora vuol dire che y avrà risposto `"YES"` alla proposta di x, e per costruzione del protocollo x, che è nello stato di `ACTIVE`, aggiungerà y nel suo vicinato. Contrariamente, se y ha risposto `"YES"` vuol dire che era nello stato di `IDLE`, e che la prima richiesta recevuta è stata quella di x. Per costruzione del protocollo, y conta x come padre e lo aggiungerà al suo vicinato. A questop punto è necessario dimostrare che l'unione di tutti i `tree-neighbors`(x) è *connessa*. Anche questo è banalmente verificabile da fatto che ogni nodo x (eccetto la sorgente) è collegato al proprio genitore tramite una sequenza di *YES-links* che risale fino alla sorgente. Perció, data una qualsiasi coppia di nodi x e y, essi saranno connessi dal cammino $x \rightsquigarrow s \rightsquigarrow y$, dove s è il nodo sorgente. Infine non resta che dimostrare che il grafo risultante del protocollo è *aciclico*. Questo si può inferire dal fatto che ogni nodo (eccetto la sorgente) invia una solo risposta `"YES"`. la quale permetterà al nodo di "appendersi" all'albero. Supponiamo per assurdo che dal nodo x esistano due cammini semplici distinti fino alla sorgente s che formano un ciclo. Anche se i due cammini possono iniziare "scendendo" di livello (raggiungendo quindi un nodo x'), dato che entrambi i cammini risalgono alla sorgente, allora certamente da un certo punto in poi passerranno per un arco che parte da un nodo figlio verso un nodo padre, salendo quindi di livello. Consideriamo solo uno dei due lati del ciclo (il ragionamento è simmetrico). Supponiamo che il primo arco del cammino che risale di livello è un certo (a, x) (o (b, x')), dove a è il genitore di x. Dato che (a, x) è il primo arco risalente, vuol dire che prima di prendere (a, x) si è scesi di livello con tutti archi del tipo $Padre \rightarrow Figlio$. Però sappiamo che ogni nodo ha un singolo nodo padre, perció il nodo a corrisponde al nodo y (corrispettivamente b corrisponde al nodo y'), e che quindi certamente si sarà preso l'arco (x, y) (o (x', y')). Questo però non può accadere in quanto nei cammini semplici si può ripercorrere due volte uno stesso arco, perció entrambi i lati dell'ipotetico ciclo devono necessariamente partire da x a risalire. Ma dato che x ha un solo padre allora esiste un solo cammino (risalente) da x alla sorgente s.

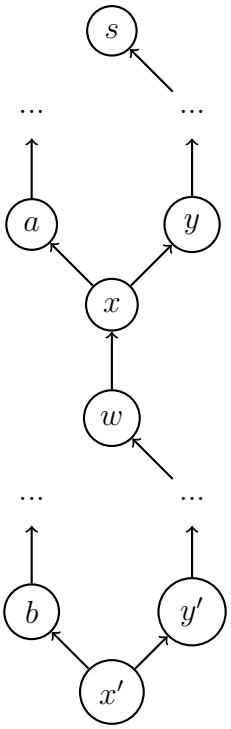


Figura 4.1: Rappresentazione grafica della dimostrazione.

4.2.3 Message complexity

Consideriamo che i messaggi che passano sugli archi sono del tipo `Q`, di richiesta di diventare figlio, e di tipo `R`, di risposta, il quale a sua volta può essere di tipo `"YES"` o `"NO"`. Si vuole quindi contare per ogni tipologia di messaggi `Q`, `"YES"` e `"NO"` quanti ne vengono trasmessi per ogni arco. Iniziamo peró considerando quali situazioni è possibile che accadono a quali no:

- Casi possibili:
  - Caso `Q`-`"YES"`, ovvero nel caso in cui su di un arco passa una proposta `Q` e poi la risposta `"YES"`.
  - Caso `Q`-`Q`, ossia quando due nodi vicini passano nello stato `ACTIVE` e si scambiano reciprocamente una proposta `Q`.
  - Caso `"NO"`-`"NO"`, ossia la diretta conseguenza del caso precedente.
- Casi impossibli:
  - Caso `"NO"`-`"YES"`: non può accadere in quanto il nodo sinistro per rispondere `"NO"` deve prima aver ricevuto una richiesta dal nodo destro, il quale avrà già un padra e quindi non puó rispondere nuovamente `"YES"`.
  - Caso `"YES"`-`"YES"`: per inviarsi reciprocamente il messaggio `"YES"`, vuol dire che entrambe le estremità dell'arco hanno inviato il messaggio `Q`, ma ciò implica che hanno già un padre e che quindi hanno già inviato `"YES"` ad un altro nodo.

Contiamo ora quanti messaggi vengono scambiati per ogni caso possbile:

- Il messaggio `Q` viene inviato nei casi `Q`-`"YES"` e `Q`-`Q`. Il caso `Q`-`"YES"` accade esattamente una volta per ogni nodo eccetto la sorgente, ovvero $n-1$ volte. Il caso `Q`-`Q` accade su tutti gli archi del vicinato di un nodo eccetto che per l'arco per il quale il nodo è diventato figlio (ovviamente escludendo la sorgente). Perciò in totale verranno scambiati $2(m-(n-1))-(n-1) = 2m-n+1$ messaggi di tipo `Q`.
- Il messaggio `"YES"` viene scambiato solamente nella situazione `Q`-`"YES"`, per un totale di $n-1$ volte.
- Il messaggio `"NO"` viene trasmesso per due volte nella situazione `"NO"`-`"NO"`, ovvero di conseguenza al caso `Q`-`Q`, per un totale di $2(m-(n-1))$.

La message complexity del protocollo `single-source-shout` sarà la somma dei precedenti valori:

$$\begin{aligned} msg(\texttt{SHOUT}) &= (2m-n+1)+(n-1)+(2m-2(n-1)) = \\ &= 4m-2n+2 = 2(2m-n+1) = 2 \cdot msg(\texttt{FLOOD}) \end{aligned}$$

4.2.4 Ottimizzazione

È possibile abbattere il fattore 2 dalla complessità del protocollo `single-source-shout` semplicemente facendo la seguente osservazione: lo scambio di messaggi di tipo `"NO"` sono in realtà superflui, infatti se un nodo x riceve una proposta `Q` da un nodo y allora certamente y è nello stato `ACTIVE`, e quindi avrà già un nodo padre, perció è inutile che x mandi una proposta ad y. Perció ricevere un messaggio `Q` può essere interpretato come la ricezione di un messaggio `"NO"` dalla stessa porta.

Esempio di codice per il protocollo `single-source-shout+`

```
class Entity:
    def __init__(self, state, message):
        self.state = state
        self.message = message

    def action(self):
        if self.state == "INITIATOR":
            spontaneously:
                self.root = True
                self.tree_neighbors = {}
                send(Q) to self.neighbors
                self.counter = 0
                self.state = "ACTIVE"
        if self.state == "IDLE":
            receiving(Q):
                self.root = False
                self.parent = Q.sender
                self.tre_neighbors = {Q.sender}
                send("YES") tp Q.sender
                self.counter = 1
                if self.counter == len(self.neighbors):
                    self.state = "DONE"
                else:
                    send(Q) to self.neighbors - {Q.sender}
                    self.state = "ACTIVE"
        if self.state == "ACTIVE"
            receiving(Q):
                self.counter += 1
                if self.counter == len(self.neighbors):
                    self.state == "DONE"
            receiving("YES"):
                self.tree_neighbors = self.tree_neighbors + {R.
    sender}
                self.counter += 1
                if self.counter == len(self.neighbors):
                    self.state == "DONE"
        if self.state == "DONE":
            None
```

Per quanto riguarda la message complexity del nuovo protocollo ottimizzato, basta osservare che non puó mai accadere la situazione `"NO"-"NO"`, ovvero quando su di un arco verrà trasmesso due volte il messaggio `"NO"`. Perció la message complexity del protocollo `single-source-shout+` sarà $msg(\texttt{SHOUT+}) = msg(\texttt{SHOUT}) - 2(m - (n-1)) = 2m$. Riguardo invece la terminazione locale del protocollo, ogni nodo sa che il suo compito è terminato nel momento in cui passa allo stato `DONE`. Sarebbe interessante sapere se

un nodo è in grado di capire quando il processo termina globalmente. Certamente se la sorgente s è in grado di sapere quando lo spanning tree è completato, allora potrebbe mandare una notifica in broadcast per avvertire tutti gli altri. Il problema è che ogni nodo ha solamente una visione locale della rete in cui si trova, perció a meno che non ci sia un nodo che riesca ad osservare lo stato della rete dall'alto mpm è possibile sapere solo col protocollo `single-source-shout` se il processo di costruzione dello spanning tree è terminato.

Osservazione: Non si può ottimizzare ulteriormente il protocollo in modo da abbattere il fattore $\Omega(m)$ semplicemente perché per costruire uno spanning tree bisogna informare in broadcast tutti i nodi della rete quantomeno che il protocollo è iniziato. Se fosse possibile farlo, allora avremmo trovato anche un protocollo di broadcast che esegue il suo task in tempo minore di m, e sappiamo che non è possibile sotto le assunzioni di link bidirezionali, unico inizializzatore, connettività e completa affidabilità.

Teorema (4.1): L'esecuzione del protocollo di broadcast genera uno spanning tree entrante della rete, ovvero dove ogni nodo possiede solamente informazioni su chi è il nodo padre, generando un albero con archi diretti "verso l'alto".

Si può osservare che lo spanning tree costruito con il protocollo `single-source-shout` (o con la sua versione ottimizzata) dipende fortemente dalla singola esecuzione e non dal protocollo, in quanto i ritardi di trasmissione variano ogni volta. Ciò implica che maneggiando adeguatamente i ritardi di trasmissione potrebbe accadere che il diametro dello spanning tree è molto più grande di quello del grafo di partenza.

4.3 Depth first traversal (DFT) protocol

Consideriamo un problema in cui un nodo iniziale possiede un *token*, e di volere che questo token (non replicabile) venga passato da tutti i nodi della rete. In pratica si tratterebbe di fare un broadcast dei token, senza però l'opportunità di duplicare il token e di inviarlo a più vicini contemporaneamente. Il protocollo `depth-first-traversal` (d'ora in poi definito `DTF`) permette di risolvere questo problema eseguendo una visita in profondità, generando così uno spanning tree della rete. L'idea del protocollo è la seguente:

- Quando un nodo x riceve per la prima volta il token da un nodo y, x deve ricordarsi che y è il suo padre dello spanning tree e poi deve inviare ad uno dei suoi vicini (eccetto y) il token, aspettando che ritorni indietro (`reply`).
- Quando un nodo x che è già stato visitato riceve di nuovo il token lo rispedisce indietro al mettente specificando che è già stato visitato, contrassegnando l'arco di ritorno come `back-edge`.

- Quando un nodo x riceve il token indietro da z, verifica se è un `back-edge` e in tal caso può affermare che z non sarà un suo vicino nello spanning tree. Se invece la risposta è solamente il token (senza specificare `back-edge`) allora x sa che z è un suo figlio nello spanning tree. Dopodichè x inoltrerà il token ad un altro dei suoi vicini.
- Quando x avrà ricevuto indietro il token da tutti i suoi vicini lo inoltrerà nuovamente al padre.

Notare che il protocollo appena descritto è l'equivalente distribuita della visita in profondità ricorsiva, in cui la visita sul nodo x termina quando sarà terminata la visita in profondità su tutti i suoi vicini (eccetto che per il nodo padre). Il protocollo terminerà nel momento in cui la sorgente s avrà ricevuto il token indietro da tutti i suoi vicini.

4.3.1 Message complexity

Possiamo osservare che esistono tre tipologie di messaggi:

- `token`, quando un nodo inoltra il token per procedere con la visita.
- `back`, quando un nodo già visitato restituisce il token al mittente.
- `return`, quando un nodo termina la propria visita e restituisce indietro il token al padre.

Inoltre su ogni arco può passare solamente uno tra i messaggi `back` e `return`, mentre il messaggio di tipo `token` passa esattamente una volta su tutti gli archi. Perció la message compelxity è esattamente $2m \in \Theta(m)$. Ricordiamo sempre che esiste un teorema che dimostra che, sotto le assunzioni in questione, non è possibile eseguire il broadcasting di un'informazione in meno di m messaggi, e dato che in questo caso sostanzialmente si sta facendo il broadcast del token, un lowerbound alla message complexity rimane $\Omega(m)$.

4.3.2 Time Complexity

Anche se il protocollo è asincrono ha senso parlare di time complexity, in quanto lo scambio di messaggi avviene in maniera *totalmente sequenziale*. Perció la time complexity è proporzionale alla message complexity.

4.3.3 Ottimizzazione

Osserviamo che in grafi molto densi una larga quantità di messaggi sarà di tipo `back`. Perció sarebbe ideale coercare di evitare che questo accada. Un'idea è quella di far inviare una notifica a tutto il vicinato di x qual'ora x riceva il token per la prima volta (ecetto che al nodo padre, ovviamente). Così facendo, tutti i vicini sapranno che x è già stato informato e non ci sarà più bisogno in futuro di ritrasmettergli il token, evitando così messaggi di tipo `back`. Il grosso vantaggio, dal punto di vista temporale, è che x può mandare la notifica ai suoi vicini in maniera *parallela*. Prima di proseguire all'inoltro del token, è necessario però che x riceva un ACK[1] dal suo vicinato, in quanto disponendo in modo adeguato i ritardi di trasmissione, potrebbe capitare che un suo vicino y riceva il

[1]Messaggio di conferma

token prima del messaggio di notifica, reinvitando così nuovamente il token a x. In tale situazione non avremmo risolto nulla, in quanto x dovrebbe rispedire il token ad y con un messaggio di `back`. Consideriamo ora i tipi messaggi che vengono scambiati:

- `token`, quando un nodo inoltra il token per procedere con la visita.
- `visited`, quando un nodo che ha appena ricevuto il token notifica i suoi vicini.
- `ack`, la risposta di avvenuta notifica da parte dei vicini.
- `return`, quando un nodo termina la propria visita e restituisce indietro il token al padre.

Ogni nodo (eccetto la sorgente s) riceverà il messaggio `token` una sola volta, e inoltre invierà il messaggio di `return` una sola volta, per un totale di $2(n-1)$ messaggi. Per quanto riguarda il messaggio di notifica `visited`, ogni nodo li invierà a tutto il suo vicinato meno che al nodo da cui hanno ricevuto il token. La sorgente sarà l'unico nodo che invierà la notifica a tutto il suo vicinato:

$$\begin{aligned} msg(\texttt{visited}) &= |N(s)| + \sum_{v \in V - \{s\}} (|N(v)| - 1) = \\ &= |N(s)| + \sum_{v \in V - \{s\}} |N(v)| - \sum_{v \in V - \{s\}} 1 = \\ &= \sum_{v \in V} |N(v)| - \sum_{v \in V - \{s\}} 1 = \\ &= 2m - (n-1) \end{aligned}$$

Inoltre per ogni notifica `visited` corrisponde una risposta `ack`, quindi la message complexity sarà:

$$\begin{aligned} msg(\text{DFT+}) &= 2(n-1) + 2 \cdot msg(\texttt{visited}) = \\ &= 2(n-1) + 2(2m - (n-1)) = \\ &= 2(n-1) + 4m - 2(n-1) = \\ &= 4m = 2 \cdot msg(\texttt{DFT}) \end{aligned}$$

Ovvero il doppio del protocollo con messaggi di tipo `back`. Come già accennato il vantaggio sia nella time complexity, infatti sappiamo che possiamo mandare i mesaggi di notifica in parallelo, e inoltre se il sistema è sincrono riceveremo i messaggi di `ack` nello stesso momento. In un sistema sincrono in cui (senza perdita di generalità) tutti i messaggi sono trasmessi in un'unità temporale, avremo che in soli due istanti (ovvero in tempo costante) ogni nodo invierà le notifiche e riceverà le risposte. Perciò il tempo necessario al completamento del task sarà:

- $n-1$ per i messaggi inviati di tipo `return`.
- $n-1$ perché ogni nodo eccetto la sorgente riceverà esattamente un messaggio `token`.
- n perché ogni nodo invia le notifiche `visited` in parallelo ai suoi vicini.
- n perché tutti i vicini di un nodo risponderanno in maniera sincrona con un `ack`.

Per un totale di $4n-2$. Con questa ottimizzazione, anche se si "paga" un fattore 2 alla message complexity otterremo che la time complexity sarà lineare in n, e non più in m. Ciò implica un miglioramento non indifferente se si considerano casi in cui la rete è molto densa, ovvero quando $m \in \Theta(n^2)$.

4.4 Multiple source spanning tree construction

Rilassiamo l'ipotesi di singolo inizializzatore e consideriamo il problema più generale in cui esiste un insieme $I \subseteq V$ di nodi inizializzatori.

Teorema (4.2): Sotto le assunzioni di link bidirezionali, connettività e totale affidabilità, il problema della costruzione di uno spanning tree con multipli inizializzatori non è deterministicamente risolvibile.

Teorema (4.2): Consideriamo una clique 3 nodi x, y, z tutti e tre inizializzatori. Consideriamo il caso sincrono in cui tutti si ativano e iniziano il protocollo al tempo t_0. Sappiamo che presi dei nodi in uno stesso stato interno $\sigma(v, t)$, allora essi da quel momento in poi si compoteranno alla stessa maniera, dato che non sono in grado di distinguere una loro identità all'interno della rete. Assodato questo fatto, allora non si potrà mai arrivare in uno stato in cui un nodo padre e gli altri due figli, in quanto questo implicherebbe che uno dei nodi si è comportato in maniera differente dagli altri due (contrariamente all'osservazione precedente).

Per risolvere questo problema è necessario risolvere un'altro problema: il problema della **leader election**. Eletto un leader tra tutti i nodi inizializzatori, esso potrà eseguire uno dei protocolli di costruzione di uno spanning tree a singola sorgente già noti.

Capitolo 5

La leader election su anelli

5.1 La definizione del problema

In un sistema distribuito in molti casi si necessita che una singola entità coordini il lavoro delle altre entità per la risoluzione di task. La presenza di tale entità può essere necessario per semplificare lo svolgimento di un task oppure perché è richiesto per la natura stessa del problema. Il problema della scelta di un leader tra un insieme omogeneo e simmetrico di individui di una popolazione è noto come il problema della leader election. Generalmente il problema di di portare la rete da una configurazione iniziale in cui tutti i nodi sono in uno stesso stato, ad una configurazione in cui tutti i nodi saranno in uno stato `follower` eccetto uno solo che dovrà essere nello stato di `leader`. Non ci sono restrizioni su quali nodi devono iniziare il task della leader election, e nemmeno su quali nodi possono diventare leader. Si può pensare alla leader election come un metodo che forza la restrizione di unico inizializzatore, infatti eletto un leader è possibile risolvere i task dei protocolli a singolo inizializzatore semplicemente ponendo il leader come nodo inizializzatore. Consideriamo il problema sotto le restrizioni di link bidirezionali, connettività e totale affidabilità.

Toerema (5.1): Sotto le restrizioni di link bidirezionali, connettività e totale affidabilità, non è deterministicamente possibile risolvere il problema della leader election.

Questo teorema forte è conseguenza della condizione di perfetta simmetria. il fenomeno della **simmetria perfetta** occorre quando una rete ha una struttura totalmente simmetrica in modo tale che ogni nodo non è in grado in nessun modo di distinguersi da un altro. Ciò significa che se tutti i nodi sono in uno stato interno σ ad uno stesso istante t, da quel momento in poi essi si comporteranno esattamente alla stessa identica maniera perché avranno la stessa visione del mondo esterno. Non potendosi comportare in maniera differente, i nodi cambieranno stato alla stessa maniera, e quindi non è possibile arrivare in una situazione in cui un nodo avrà uno stato diverso dagli altri (il `leader`). Il problema della leader election è anche noto come **rottura della simmetria**, perché appunto cerca di "rompere" tale simmetria tra i nodi. Il teorema sopra ci dice che però che per rompere la simmetria, le restrizioni citate sopra non sono sufficienti. È necessario aggiungtere la restrizione di **identificatori univoci** o **valori iniziali distinti** (in breve **ID**), nella quale si assume che ogni nodo x è *univocamente identificato* dal valore $id(x)$,

ovvero che rispetta la seguente proprietà: $\forall x, y \in V, id(x) \neq id(y)$. Notiamo che risolvere il task dell'elezione di un leader è risolvibile con la restrizione di identificatori univoci: per esempio si potrebbe eleggere come leader il nodo con *etichetta minima* (o *massima*).

5.2 Minimum finding su di un anello

Come accennato, trovare il nodo con etichetta minima può essere sfruttato per l'elezione di un leader. Consideriamo una rete ad anello, ovvero una tra le tipologie più sparse di grafo simmetrico connesso. Un anello è un grafo $R := (\{x_1, x_2, ..., x_n\}, E)$ in cui l'insieme degli archi E è definito come $E := \{(x_i, x_{i+1}) | i \in [1, n-1]\} \cup \{(x_n, x_1)\}$.

Osservazione: Non è detto che gli ID dei nodi siano globalmente consistenti, ovvero non è detto che $id(x_i) = i$, oppure che se $id(x_i) = k$ allora per forza deve essere vero che $id(x_{i+1}) = k + 1$ e $id(x_{i-1}) = k - 1$.

Osservazione: In un anello con n nodi ci sono esattamente n archi.

Assumiamo infine che ogni nodo condivide una stessa idea di destra e sinistra, ovvero se definiamo che il nodo destro di x_2 è x_3, allora il nodo destro di ogni nodo x_i sarà il nodo con indice $i + 1 \mod n$.

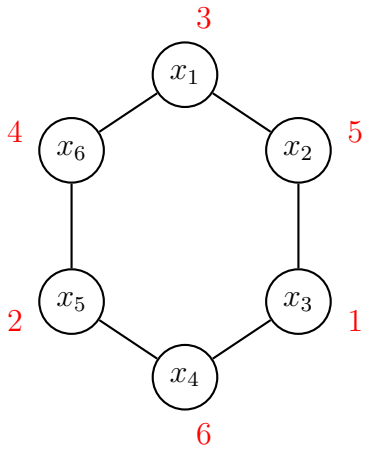


Figura 5.1: Esempio di una rete ad anello con 6 nodi (in rosso sono indicate gli ID).

5.3 Il protocollo `all-the-way`

Una prima idea abbastanza semplice è la seguente:

- Quando un nodo si attiva e inizia il protocollo inoltra un messaggio in una direzione concorde a tutti (diciamo a destra). Nel messaggio inviato sarà scritta la propria etichetta.

- Quando un nodo riceve un messaggio conserva l'etichetta scritta, tenendo traccia dell'etichetta minima ricevuta fino a quel momento. Dopodiché inoltra in avanti il messaggio (`all-the-way`).
- Quando un nodo riceve da sinistra un messaggio con la sua stessa etichetta allora vuol dire che avrà fatto il giro, e che tutti i nodi avranno ricevuto la sua etichetta, perciò non dovrà più inoltrarla.

È importante evidenziare un problema: affermare di aver considerato tutte le etichette quando la propria etichetta avrà fatto il giro è sbagliato, questo perché si necessiterebbe di un'ulteriore restrizione, ossia la **message ordering**. Se i messaggi seguissero una coda FIFO di trasmissione, allora certamente ad un nodo x tornerebbe indietro il suo messaggio non prima di aver già ricevuto gli altri $n-1$. Si potrebbe pensare allora di terminare il protocollo quando un nodo riceve esattamente n (compreso il suo di ritorno), però questo non è possibile a meno che non si conosca a priori in valore di n. È però possibile ricavare n in qualche maniera. Sappiamo che se il messaggio di x torna a destinazione allora certamente sarà stato inoltrato da n nodi (compreso x). Se inseriamo un *contatore* nel messaggio che viene incrementato ad ogni inoltro, allora il valore del contatore quando il messaggio tornerà a destinazione sarà esattamente pari ad n. Perciò quando x riceve il suo messaggio da sinistra potrà ricavare il valore di n e comportarsi in due modi:

- Se x ha già ricevuto n etichette allora basta fare il minimo tra tutte per identificare l'eticehtta minima dell'intera rete.
- Se x ha ricevuto $k < n$ etichette, allora dovrà attendere di ricevere altre $n-k$ *nuove* etichette, per poi poter calcolare il minimo.

In questa maniera ogni nodo potrà sempre terminare localmente, riuscendo ad identificare un `leader` globale comune a tutti. Definiami l'insieme degli stati:

- $S := \{\texttt{ASLEEP}, \texttt{AWAKE}, \texttt{FOLLOWER}, \texttt{LEADER}\}$.
- $S_{init} := \{\texttt{ASLEEP}\}$.
- $S_{final} := \{\texttt{FOLLOWER}, \texttt{LEADER}\}$.

Esempio di codice per il protocollo `all-the-way`

```
class Entity:
    def __init__(self, state, id):
        self.state = state
        self.id = id
        self.ring_size = None

    def initialize(self):
        self.count = 0
        self.size = 1
        self.known = False
        self.min = self.id
        send_right({msg: "ELECTION", id: self.id, count: 1})

    def check(self):
        assert self.size == ring_size
        self.state = "LEADER" if self.min == self.id else "FOLLOWER"

    def action(self):
        if self.state == "ASLEEP":
            spontaneously:
                self.initialize()
                self.state = "AWAKE"
            receiving(msg):
                self.initialize()
                self.count += 1
                self.min = min(self.min, msg.id)
                self.state = "AWAKE"
        elif self.state == "AWAKE":
            receiving(msg):
                if self.id == msg.id:
                    ring_size = msg.count
                    self.known = True
                    self.check()
                elif self.id != msg.id:
                    send_right({msg: msg.msg, id: msg.id, count: msg
    .count+1})
                    self.count += 1
                    self.min = min(self.min, msg.id)
                    if self.known:
                        self.check()
```

5.3.1 Correttezza e complessità

È facilmente intuibile la correttezza del protocollo, in quando l'assunzione di totale affidabilità ci garantisce che ogni nodo prima o poi riceverà le informazioni riguardo le etichette di tutti quanti, riusciendo poi a calcolarne il minimo. Per quanto riguarda la message complexity basta osservare che ogni etichetta compierà l'intero giro dell'anello, per un totale di:

$$msg(\texttt{ALL-THE-WAY}) = n^2$$

Per la time complexity invecepossiamo solo fare affermazioni riguardo un *tempo ideale*. Assumendo che il sistema di sincrono, che tutti i nodi iniziano il protocollo ad uno stesso istante t e che il tempo di trasmissione è di una sola unità fissata, allora il tempo ideale è n. Il caso peggiore invece è quando si attiva un solo nodo x_1 al tempo t_0. Inviando il messaggio al suo vicino x_2, esso si attiverà al tempo $t_1 = t_0 + 1$. Il nodo x_3 si attiverà al tempo successivo, e così via fino al nodo x_N che si attivertà dopo $n - 1$ istanti. Da quel momento in poi, l'ultimo nodo che terminerà il processo sarà proprio x_n, il quale dovrà aspettare che la sua atichetta faccia il giro completo dell'anello, risultando così in una time complexity di:

$$time(\texttt{ALL-THE-WAY}) \leq 2n - 1$$

Infine è possibile calcolare una **bit complexity** del protocollo `all-the-way`. Trascurando la stringa `"ELECTION"` che può anche essere omessa, è sufficiente contare il numero di bit necessari per rappresentare l'etichetta $id(x)$ e il contatore che viene mandato. Considerando il messaggio `id+cound` (dove l'operatore + sta per la *concatenazione* dei due valori), il numero massimo di bit necessari per rappresentare un messaggio se lo si codifica in binario sarà $\log_2(id(x) + n)$, con una complessità globale di $\Theta(n^2 \log(id(x) + n))$.

5.4 Il protocollo `as-far (as-it-can)`

Un'idea per ottimizzare il processo di ricerca dell'etichetta minima è quella che non è necessario inoltrare un'etichetta che so già che non potrà essere minima. Perció, quando un nodo x riceve un'etichetta che è maggiore di quella che x momentaneamente ritiene minima, semplicemente non la inoltrerà. Un'etichetta viaggia in avanti *più a lungo che può* (*as far as it can*) e non *in ogni nodo* (*all the way*). Certamente il nodo con etichetta minima sarà l'unico a veder ricevere dal lato opposto la propria etichetta, perció potrà terminare *localmente* il suo compito. Per terminare *globalmente*, il nodo che alla fine potrà stabilire di essere quello con etichetta minima dovrà necessariamente notificare (in broadcast) gli altri nodi. Osservare che il vosto aggiuntivo in bit complexity non è eccessivo, in quando tutti i nodi certamente convergeranno su una singola etichettà minima, e quindi basterà un solo bit per terminare il task.

5.4.1 Message complexity

Il caso peggiore in termini di message complexity è quando le etichette seguono un ordine crescente di valori nel senso in cui il messaggio viene inoltrato. Nel caso peggiore inizierà il protocollo con etichetta n, ma il suo messaggio verrà bloccato dal nodo 1. Dopodichè si attiva il nodo $n - 1$, la cui etichetta verrà bloccata dopo 2 passi sempre dal nodo 1. E così via fino al nodo 2 la cui etichetta farà quasi un giro completo dell'anello fino a bloccarsi sempre al nodo 1. Infine si sevglia il nodo 1 la cui etichetta farà il giro

completo dell'anello, scatenando così il broadcast di notifica di fine protocollo. La message complexity in questa situazione critica sarà:

$$msg(\texttt{AS-FAR}) \leq n + \sum_{i=1}^{n} i = n + \frac{n(n-1)}{2}$$

Questa complessità non è asintoticamente migliore rispetto a quella del protocollo `all-the-way`. Banalmente invece il caso migliore è quando il nodo con etichetta minima inizia il protocollo per primo e la sua etichetta riesce a fare il giro prima che le altre vengano inoltrate. Ricapitolando la complessità temporale del protocollo ha un upper-bound di $O(n^2)$ e un lower-bound di $O(n)$

5.5 Le distanze controllate - la tecnica degli stage

Osserviamo che sia nel protocollo `as-far` che nel protocollo `all-the-way` le etichette viaggiano lungo l'anello in maniera incontrollata. In particolare nel protocollo `as-far` viaggiano finché non vengono filtrati da un nodo con etichetta più bassa, mentre nel protocollo `all-the-way` addirittura viaggiano lungo l'intero anello. L'idea sarebbe quindi quella di controllare la trasmissione di una etichetta ad una distanza limitata, e di avere dei feedback per eventualmente bloccare o estendere la trasmissione di un'etichetta. Più precisamente un nodo attivo x esegue il protocollo a stages, dove nello stage i-esimo esso inoltrerà a destra e a sinistra la sua etichetta $id(x)$ fino ad un massimo di distanza 2^{i-1}. Così facendo l'etichetta avrà ricoperto un numero di nodi pari a 2^i, ovvero 2^{i-1} nodi a destra e 2^{i-1} nodi a sinistra. Quando un nodo y riceve l'etichetta di x la inoltrerà solamente se $id(y) > id(x)$. Nel caso in cui y inolterrà $id(x)$ allora vorrà dire che certamente y non sarà mai eletto leader, e quindi passerà in uno stato di `defeated` in cui potrà solamente inoltrare le etichette dagli altri. Se l'etichetta $id(x)$ arriva a distanza 2^{i-1} allora essa verrà rispedita indietro come messaggio di feedback. Se il nodo x riceverà un feedback da entrambi i lati vorrà dire che tutti i nodi a distanza 2^{i-1} da lui avranno etichette maggiori, e quindi dovrà inoltrare l'etichetta più lontano. In questo caso decidiamo che x è sopravvissuto allo stage i e quindi potrà passare allo stage $i+1$. Notiamo che se x non riceve il feedback da uno o tutti e due i lati, vuol dire che esiste almeno un nodo z entro 2^{i-1} passi con etichetta minore $id(z) < id(x)$, e quindi prima o poi x verrà battuto dall'etichetta $id(z)$. Infine, quando un nodo x^* riceverà da destra il messaggio che aveva inviato a sinistra, oppure da sinistra quello che aveva inviato a destra, allora x^* potrà affermare di essere il leader. A questo punto x^* manderà in broadcast un messaggio di notifica, che farà terminare globalmente il protocollo, facendo passare tutti i nodi nello stato `defeated` allo stato `follower`. Ricapitolando:

- In ogni **stage** i c'è un insieme di nodi candidati al ruolo di leader.
- Ogni candidato manda la sua etichetta a destra e a sinistra.
- L'etichetta viaggia finché non incontra un nodo con etichetta più piccola oppure finché non arriva a distanza 2^{i-1}.
- Se l'etichetta non incontra nessun nodo con etichetta più piccola allora essa tornerà indietro.

- Se un nodo riceverà indietro le etichette da entrambi i lati allora potrà passare allo stage successivo.

In aggiunto ci sono altre tre meta-regole:

- Se un nodo nello stato `candidate` riceverà un'etichetta dal lato opposto a quello da cui l'ha mandata, allora diventerà `leader` e lo notificherà ai suoi vicini.
- Se un nodo `candidate` riceverà un'etichetta minore della sua allora passerà allo stato `defeated`.
- Un nodo nello stato `defeated` potrà solamente inoltrare i messaggi di altri nodi, e passerà allo stato `follower` quando riceverà la notifica del leader.

Definiamo ora formalmente gli stati del protocollo e una sua possibile implementazione:

- $S := \{\texttt{ASLEEP}, \texttt{CANDIDATE}, \texttt{DEFEATED}, \texttt{FOLLOWER}, \texttt{LEADER}\}$.
- $S_{init} := \{\texttt{ASLEEP}\}$.
- $S_{final} := \{\texttt{FOLLOWER}, \texttt{LEADER}\}$.

Esempio di codice per il protocollo `stage-technique`

```
class Enitity:
    def __init__(self, state, id):
        self.state = state
        self.id = id

    def limit(i):
        assert i > 0
        return 2**(i+1)

    def initialize(self):
        self.stage = 1
        self.limit = limit(1)
        self.count = 0
        \* 1 = right; 0 = left *\
        send("FORTH", self.id, self.stage, self.limit, direction=1)
    to self.neigh(direction=1)
        send("FORTH", self.id, self.stage, self.limit, direction=0)
    to self.neigh(direction=0)

    def process(self, msg):
        msg.limit -= 1
        if msg.limit == 0:
            self.send("BACK", msg.id, direction = not msg.direction)
     to msg.sender
        else:
            self.send("FORTH", msg.id, msg.state, msg.limit) to self
```

```
    .neigh(direction=msg.direction)
24
25      def notify(self):
26          send("NOTIFY", leader=self.id, direction=1)
27
28      def check(self):
29          self.count += 1
30          if self.count == 2:
31          self.count = 0
32          self.stage += 1
33          self.limit = limit(self.stage)
34          \* 1 = right; 0 = left *\
35          send("FORTH", self.id, self.stage, self.limit, direction=1)
    to self.neigh(direction=1)
36          send("FORTH", self.id, self.stage, self.limit, direction=0)
    to self.neigh(direction=0)
37
38      LEADER = None
39
40      if self.state == "ASLEEP":
41          spontaneously:
42              self.initialize()
43              self.state = "CANDIDATE"
44          receiving(msg):
45              if msg.id < self.id:
46                 self.process(msg)
47                 self.state = "DEFEATED"
48              else:
49                  self.initialize()
50                  self.state = "CANDIDATE"
51      elif self.state == "CANDIDATE":
52          receiving(msg):
53              if msg.type == "FORTH":
54                  if msg.id < self.id:
55                      self.process(msg)
56                      self.state = "DEFEATED"
57                  elif msg.id == self.id:
58                      LEADER = self.id
59                      self.notify()
60              if msg.type == "BACK":
61                  if msg.id == self.id:
62                  self.check()
63              if msg.type == "NOTIFY":
64                  LEADER = msg.id
65                  send(msg) to self.neigh(direction=msg.direction)
66                  self.state = "FOLLOWER"
67      elif self.state == "DEFEATED":
```

```
68          receiving(msg):
69              send(msg) to self.neigh(direction=msg.direction)
70              self.state = "FOLLOWER"
```

5.5.1 Correttezza

La correttezza del protocollo è implicata dalla descrizione del protocollo. Infatti solamente l'etichetta minima sarà in grado di compiere un giro completo, portando allo stato `defeated` tutti gli altri nodi dell'anello. Inoltre tramite il messaggio di notifica da parte del leader tutti gli altri nodi passeranno allo stato `follower`. In conclusione si arriverà sempre allo stato finale accettabile in cui si ha un solo leader e tutti followers.

5.5.2 Message complexity

Considerando che il ritardo dei messaggi è impredicibile si osserverà che ad un dato tempo t non solo ci saranno nodi in stati differenti (come ci si aspetta) ma anche che ci saranno nodi candidati in fasi differenti. Più in generale, non essendoci un ordinamento dei messaggi, può capitare che dei messaggi sorpassino degli altri. Perció, per i motivi precedentemente citati, faremo un'analisi procedendo per *stage logici*, ovvero non considerando che differenti nodi possono eseguire uno stato stage a tempi differenti. Sia $i > 1$, definiamo n_i il numero di nodi che iniziano lo stage i, ovvero quelli sopravvissuti allo stage $i-1$. Per essere sopravvissuti allo stage $i-1$ (e quindi ritrovarsi allo stage i) vuol dire che il nodo x ha un'etichetta $id(x)$ minore di 2^{i-2} nodi alla sua destra e 2^{i-2} nodi alla sua sinistra. Perció possiamo dire che fase $i-1$ può sopravvivere al più un solo nodo ogni $2^{i-2}+1$, e quindi avremo che $n_i < \frac{n}{2^{i-2}+1}$. Non ci può essere più di un sopravvissuto ogni $2^{i-2}+1$ nodi in quanto se ce ne fossero di più, per esempio x e y, vorebbe dire che x e y si sono scambiati le etichette senza però che uno dei due venisse "sconfitto" e ciò implicherebbe che $id(x) < id(y) < id(x)$, il che è un assurdo. Allo stage i ogni nodo candidato x inoltra la sua richiesta ad al più $2 \cdot 2^{i-1} = 2^i$ nodi, per un totale massimo di messaggi di tipo `FORTH` pari a $n_i 2^i$. Dato che alla fase i sopravviveranno n_{i+1} candidati, e per ognuno verranno necessariamente trasmessi altri $2 \cdot 2^{i-1} = 2^i$ messaggi di feedback, per un complessivo massimo di $n_{i+1}2^i$ messaggi di tipo `BACK`. Infine ogni nodo che non sopravvive alla fase i riceverà nessuno o al più un feedback, per un massimo di 2^{i-1} ulteriori messaggi. Dato che sono $n_i - n_{i+1}$ nodi `defeated` nella fase i, si pagherà un costo in messaggi al più $(n_i - n_{i+1})2^{i-1}$ messaggi ulteriori. Perciò complessivamente nella fase $i > 1$ verranno mandati al più il seguente valore di messaggi:

$$
\begin{aligned}
2n_i 2^{i-1} + 2n_{i+1}2^{i-1} + (n_i - n_{i+1})2^{i-1} &= (3n_i + n_{i+1})2^{i-1} \leq \\
&\leq (3\lfloor \frac{n}{2^{i-2}+1} \rfloor + \lfloor \frac{n}{2^{i-1}+1} \rfloor)2^{i-1} < \\
&< (3\lfloor \frac{n}{2^{i-2}+1} \rfloor + \lfloor \frac{n}{2^{i-1}+1} \rfloor)2^{i-1} < \\
&< (3\lfloor \frac{n}{2^{i-2}} \rfloor + \lfloor \frac{n}{2^{i-1}} \rfloor)2^{i-1} = \\
&= 6n + n = 7n
\end{aligned}
$$

Dando una stima di $n_i \leq n$ avremo che le fasi potranno essere al più $\log_2 n$. Complessivamente la message complexity sarà:

$$\begin{aligned} msg(\texttt{STAGE}) &\leq \overbrace{O(n)}^{\text{Stage 1}} + \sum_{i=2}^{\log_2 n} 7n < \\ &< O(n) + n \sum_{i=2}^{\log_2 n} 7 = \\ &= O(n) + 7n \log_2 n \in O(n \log n) \end{aligned}$$

Capitolo 6

Il broadcast su reti radio

6.1 Il modello rete radio

Una **rete radio** è un insieme di dispositivi fisii collocati in uno spazio euclideo, ognuno dei quali è dotato di un **trasmettitore radio** (o **wireless**). Ogni nodo v della rete ha un fissato **raggio di trasmissione** $r(v) > 0$, entro il quale potrà trasmettere il suo segnale. Un qualsiasi nodo u per ricevere un messaggio da $v$deve essere ad una distanza minore del raggio di trasmissione $r(v)$, ossia $||v - u|| \leq r(v)$. Inoltre il nodo v non può selezionare a quale dei suoi nodi vicini inviare il messaggio e a quali no: il messaggio verrà trasmesso in *broadcast* a tutti i nodi entro il raggio di trasmissione di v. Questo tipo di comunicazione è detta **basata sui nodi**. Data quindi una rete radio si può costruire un **grafo diretto delle comunicazioni**, in cui esiste l'arco diretto (u, v) solo se v è coperto dal raggio di trasmissione di u.

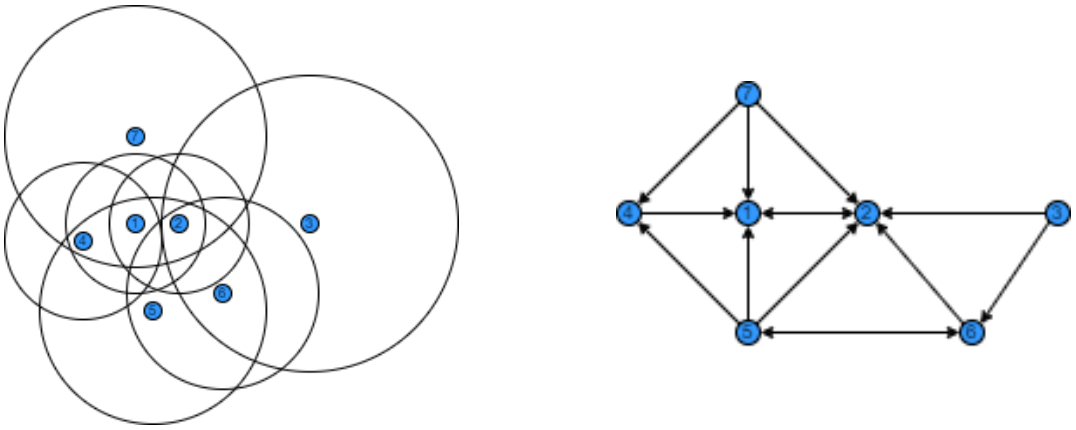


Figura 6.1: Un esempio di rete radio e del corrispettivo grafo di connessione.

Le reti radio sono anche dette *sconosciute* in quanto i nodi non hanno nessuna conoscenza in merito agli altri nodi e alla loro posizione, perciò è come considerare un grafo in cui tutti i nodi conoscono il loro vicinato, in cui hanno una sola porta dalla quale trasmettere il messaggio e basta. Infine le reti radio sono considerate essere un sistema completamente **sincrono**. Tuti i nodi condividono un **clock globabe**, perciò diremo che i nodi agiscono in **slot temporali**. Inoltre si assume che i messaggi vengono trasmessi nella propria area di trasmissione in un solo slot temporale.

6.2 Il modello d'interferenza

In un contesto reale, in genere, se troppi dispositivi trasmettono nello stesso momento e in una stesa area si crea troppo *rumore* o *interferenza*. In una rete radio quindi, se un nodo v riceve più di un messaggio in uno stesso slot temporale, assumeremo che si crei un'interferenza e che quindi v non ha recevuto il messaggio. Più formalmente diremo che il nodo v viene informato allo slot temporale $t > 0$ se e solo se esiste un unico suo vicino che gli trasmette il messaggio al tempo t.

6.3 Il broadcast su reti radio

Una delle task più comuni in un ambiente reale è quello del broadcast di un segnale su una rete radio. Come accennato, questáttività soffre del problema dell'interferenza, perciò si vuole trovare un protocollo che porti a termine il broadcast di un'informazione considerando questa ulteriore difficoltà. Modelliamo quindi il problema considerando come grafo il grafo di communicazione G derivato dalla rete radio, assumendo di trovarsi sotto le seguenti assunzioni:

- *Sistema sincrono*: ogni nodo condivide uno stesso clock globale che scandisce le azioni.
- *Unica sorgente* s che avvia il protocollo.
- *Connetività* di G, non necessariamente forte, ma quantomeno ogni nodo deve essere raggiungibile dalla sorgente s.
- *Presenza di interferenze*: un nodo v viene informato al tempo t se esiste un unico nodo che al tempo $t - 1$ gli trasmette il mesaggio.

Come primo passo potremmo iniziare considerando il protocollo di flooding. È facile trovare un controesempio in cui il protocollo non informerà mai tutti i nodi

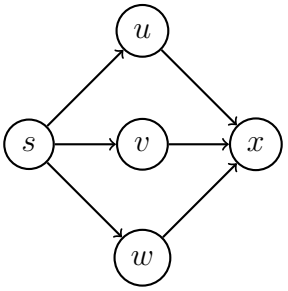


Figura 6.2: Controesempio in cui il protocollo flooding non funziona.

Consideriamo il grafo di comunicazione nella precedente immagine, e supponiamo che la sorgente sia il nodo s. Allo slot temporale t_0 la sorgente s invierà il messaggio ai noi u, v e w, e allo slot temporale successivo i nodi u, v e w invieranno il messaggio al nodo x, causando un'interferenza. Dopo che u, v e w avranno trasmesso, essi entreranno nello stato `done` e non trasmetteranno mai più. Così facendo il nodo x non riceverà mai il messaggio, e il task non sarà mai risolto.

6.4 Il protocollo `round-robin`

Assumiamo che i nodi univocamente identificati da un valore nell'intervallo $[1, n]$, e che tutti i nodi conoscano una buona approssimazione di n. Per semplicità al momento assumiamo che conoscevano il valore esatto di n. L'idea del protocollo `round-robin` (in breve `RR`) è quella di scandire il protocollo in fasi con durata complessiva di n slot temporali ciascuno. Dato che ogni fase $k \geq 0$ dura esattamente n istanti, ogni nodo con etichetta i trasmetterà all'i-esimo slot temporale della fase k. Più formalmente per ogni $k \geq 0$, il nodo i può trasmettere solo negli istanti $nk + i$. È importante osservare che è possibile definire una scansione temporale delle azioni da svolgere grazie all'assunzione di sincronismo del sistema. Questo sistema ci garantisce che non è possibile che accada una situazione d'interferenza, in quanto può "parlare" (trasmettere) un solo nodo alla volta.

Esempio di codice per il protocollo `round-robin`

```
class Enitity:
    def __init__(self, state, id, msg):
        self.state = state
        self.id = id
        self.msg = msg

    def action(self):
        if self.state == "SOURCE":
            spontaneously:
                self.state = "INFORMED"
        elif self.state == "ASLEEP":
            receiving(msg):
                self.state = "INFORMED"
        elif self.state == "INFORMED":
            global clock, n
            if clock % n == self.id:
                transimt(msg)
                self.state = "DONE"
        elif self.state == "DONE":
            None
```

6.4.1 Correttezza

Teorema (6.1): Nel protocollo `round-robin` le precedenti assunzioni, tutti i nodi a distanza k dalla sorgente s verranno informati entro k fasi.

Teorema (6.1): La dimostrazione di questo teorema verrà fatta per induzione sulla distanza k. Il passo base è abbastanza semplice, in quanto nella prima fase l'unico

nodo a trasmettere sarà la sorgente s nello slot temporale $id(s)$. Supponiamo quindi per ipotesi induttiva che sia vero che per ogni fase $k > 0$ ogni nodo nell'insieme L_k (ovvero i nodi a distanza esattamente k dalla sorgente) verrà informato entro la fine della fase k. Consideriamo ora un qualsiasi nodo $w \in L_{k+1}$. Per definizione di L_{k+1}, esiste almeno un nodo in $v \in L_k$ tale che esista l'arco diretto (v, w). Per ipotesi induttiva, sappiamo che entro la fine della fase k il nodo v è stato informato. Se v ha ricevuto il messaggio in uno slot temporale $kn + i < kn + id(v)$, allora v può trasmettere durante la fase k stessa al tempo $kn + id(v)$, altrimenti se ha ricevuto il messaggio ad uno slot temporale $kn+i > kn+id(v)$, allora v trasmetterà certamente nella fase $k + 1$, più precisamente al time slot $(k + 1)n + id(v)$. In ogni caso, per definizione del protocollo `round-robin`, sia che v trasmetta al tempo $kn + id(v)$ sia che trasmetta al tempo $(k + 1)n + id(v)$, sarà l'unico che in quel momento invierà il messaggio a w. Perciò possiamo affermare che w riceverà almeno una volta il messaggio senza alcuna interferenza, e questo vale per ogni $w \in L_{k+1}$. Infine, per ipotesi di connettività di G, sappiamo che per ogni nodo $x \in V$ esiste un intero h tale che $x \in L_h$.

6.4.2 Terminazione e time complexity

Sia D l'eccentricità della sorgente s, ovvero il più lungo dei cammini minimi che partono da s. Sicuramente in D fasi tutti i nodi della rete verranno informati grazie al protocollo `round-robin`. Il problema è che nessun nodo conosce né l'eccentricità di s né il diametro della rete. Però è noto che in una rete connessa di n nodi il diametro è al più $n - 1$. Perciò, dato che ogni nodo conosce il valore n, ognuno può stabilire che entro al più $n - 1$ fasi tutti i nodi saranno informati, e quindi potranno terminare l'esecuzione del protocollo. Segue quindi il seguente teorema:

Teorema (6.2): Sia la rete G, con n nodi e sorgente s, avente eccentricità D. Sotto le precedenti assunzioni il protocollo `round-robin` completerà il suo task in esattamente $nD \in O(n^2)$ slot temporali e tutti i nodi termineranno 'esecuzione del protocollo in $O(n^2)$ slot temporali.

6.4.3 Message complexity

La message complexity di questo protocollo è banale, in quanto ogni nodo trasmetterà lungo tutti i suoi archi uscenti una volta sola il messaggio, perciò la message complexity sarà pari al numero di archi del grafo di trasmissione, ossia $msg(\texttt{RR}) = |E|$.

6.5 La generalizzazione del problema

Fin ora abbiamo assunto che tutti i nodi hanno una conoscenza di n, oppure una sua buona approssimazione. Per esempio, se tutti i nodi avessero come approssimazione n un valore del tipo $3.5n$, l'efficienza del protocollo `round-robin` in termini asintotici non cambia. Supponiamo ora invece che i nodi abbiano un'approssimazione N del numero di nodi n. Col protocollo `round-robin` il tempo di terminazione globale sarà $O(N^2)$, e

se $N \cong 2^n$ il tempo di completamente sarà $O(2^{2n})$. Supponiamo in maniera ancor più generale che i nodi della rete non conoscano affatto n né una sua stima. Generalmente, in questi casi in cui non si conoscono alcuni parametri importanti per il problema, si ricorre alla tecnica della **simulazione**, in cui si provano tutti i possibili valori che i parametri mancanti possono asumere. In questo caso quinid, dato che stiamo in un sistema totalmente sincrono, si eseguirà per prima cosa il protocollo `round-robin` supponendo che n sia pari ad 1, poi si ripete supponendo che sia uguale a 2, e così via. Così facendo, esisterà un tempo t nel quale verrà eseguito il protocollo col corretto valore di n, portando a termine così il task del broadcast su reti radio. Importante specificare però che qualora un nodo abbia un'etichetta x maggiore del corrente valore di n, esso non dovrà mai essere in grado di mandare il messaggio, in quanto potrebbe esistere un altro nodo con etichetta $id(y) \leq id(x)$ tale che $id(y) \equiv_n id(x)$ e quindi potrebbero occorrere interferenze. In questo caso il task verrà completato in un tempo dell'ordine di":

$$time(\texttt{RR}) = \sum_{i=1}^{n} time(\texttt{round-robin}(i)) = \sum_{i=1}^{n} \Theta(i^2) \in \Theta(n^3)$$

6.5.1 Terminazione locale

Consideriamo il nodo v_j con etichetta $id(v_j) = j$, e consideriamo l'esecuzione del protocollo di `round-robin` di parametro $n = i$ in cui v_j viene informato. Se abbiamo che $j < i$ sappiamo che se siamo fortunati v_j trasmetterà il messaggio direttamente nella fase i, altrimenti dovrà attendere la fase successiva $i + 1$. Se invece abbiamo che $j > i$, allora sappiamo che v_j non parlerà mai nella i-esima simulazione del protocollo `round-robin`. Esisterà comunque un tempo $i^* \geq j$ in cui v_j avrà opportunità di trasmettere il messaggio ai suoi vicini. In entrambi i casi, per come è costruito il protocollo, sappiamo che quando v_j trasmetterà non ci potranno essere altri nodi che trasmetteranno nello stesso istante. Perciò v_j informerà tutti i suoi vicini senza interferenze, e potrà concludere localmente il suo compito, passando allo stato `done`.

6.5.2 Ottimizzazione

Un'ottimizzazione abbastanza intuitiva è quella di non far crescere il valore simulato di n in maniera lineare, bensì in maniera esponenziale. Ovvero, iteriamo le simulazioni del protocollo `round-robin` ponendo il parametro $n = 2^i$ per ogni $i > 0$. Per quanto riguarda il completamente del task, esisterà certamente un certo i tale che $2^{i-1} < n \leq 2^i$, garantendo quindi che entro 2^i simulazioni tutti i nodi verranno informati. Per il tempo di completamente invece, basta osservare che ci saranno al più $\lceil \log_2 n \rceil$ simulazioni, ognuna delle quali con complessità temporale di $O(n^2)$. Perciò avremo che:

$$time(\texttt{RR}) = \sum_{i=1}^{\lceil \log_2 n \rceil} time(\texttt{round-robin}(2^i)) = O(n^2 \log n) \in O(n^3)$$

Parte II

Protocolli randomizzati distribuiti

Capitolo 7

Introduzione

7.1 Algoritmo probabilistico

Un **algoritmo probabilistico** $\mathcal{A}$ è un algoritmo che effettua accessi ad una fonte $\mathcal{S}$ di random bits, e che prende decisioni in accordo ai bit casuali dati dalla fone $\mathcal{S}$. Sia un qualsiasi input x di $\mathcal{A}$ di dimensione $|x|= n$, e sia $r = r(n)$ il numero totale di random bit che $\mathcal{A}$ richiede ad $\mathcal{S}$ durante l'esecuzione di $\mathcal{A}(x)$. Avremo quindi che la correttezza del risultato dell'esecuzione di $\mathcal{A}(x)$ e il suo tempo di completamento sono **variabili aleatorie** sopra lo spazio di probabilità $\Omega = (\{0,1\}^r, \mathcal{P}(\cdot))$, dove $\mathcal{P}(\cdot)$ è la **distribuzione di probabilità** indotta da $\mathcal{A}(x)$. Osserviamo che se $\mathcal{A}$ sceglie gli r bits *uniformemente a caso* e in maniera *indipendente*, allora avremo che la distribuzione $\mathcal{P}(\cdot)$ è esplicitamente definita come $\forall y \in \{0,1\}^r, \mathcal{P}(y) = 2^{-r}$. Infine, per semplicità assumiamo che $\mathcal{A}$ è un algoritmo per il *problema decisionale* Π. Diamo ora una definizione di concetto di **probabilità d'errore** di $\mathcal{A}$ in termini di eventi nello spazio di probabilità $\sum$, e quindi una migliroe definizione di algoritmo probabilistico.

Teorema (7.1): Un algoritmo $\mathcal{A}$ per un problema Π ha una **probabilità d'errore** $\sigma \in [0,1]$ se, perogni possibile input $x \in \{0,1\}^n$, avremo che:

$$\mathcal{P}_\Omega(\mathcal{A}(x) = \Pi(x)) \geq 1 - e$$

Teorema (7.1): Un algoritmo $\mathcal{A}$ risolve Π **con alta probabilità** se, per valori di n sufficientemente grandi e per ogni input $x \in \{0,1\}^n$ avremo che per qualche costante $\beta > 0$ è vero che:

$$\mathcal{P}_\Omega(\mathcal{A}(x) = \Pi(x)) \geq 1 - n^{-\beta}$$

A questo punto possiamo definire in maniera del tutto analoga il concetto di **protocollo distribuito probabilistico**. Sia quindi un dato protocollo $\mathcal{A}$ che viene eseguito su un grafo $G = (V, E)$ con $|V|-n$ nodi e che inizia con una configurazione inziale x. Ogni nodo v ha accesso ad una fonte *privata* e *indipendente* $s(v)$ di random bits. Le decisioni di ogni nodo v in accordo in quanto descritto dal protocollo $\mathcal{A}$, dipendono anche dalle sequenze di random bits generate dalle fonti di casualità $s(v)$. Supponiamo che il massimo numero di random bits generati dalle fonti durante l'esecuzione di $\mathcal{A}(G, x)$

sia una certa quantità $r > 0$. Perciò il risultato dell'esecuzione $\mathcal{A}(G, x)$ (o il suo tempo di completamento) è una variabile aleatoria completamente determinata dallo spazio di probabilità $\Omega - (\{0,1\}^{n \times r}, \mathcal{P}(\cdot))$, dove $\mathcal{P}(\cdot)$ è la distribuzione di probabilità indotta da $\mathcal{A}(G, x)$. Analogamente possiamo estendere i concetti di probabilità di errore e con alta probabilità anche per i protocolli destribuiti probabilistici.

7.2 La leader election su anelli non etichettati

Abbiamo visto nello studio della leader election che non è possibile risolvere deterministicamente tale problema senza l'assunzione che tutti i nodi abbiano un *ID unico*. Assumiamo ora un contesto in cui abbiamo un rete $G = (V, E)$ di n nodi *anonimi* (senza un ID unico) in cui però tutti i nodi conoscono n e sanno di essere in un anello, e supponiamo inoltre che ogni nodo v abbia accesso ad una fonte privata e indipendente di casualità $s(v)$. È possibile definire un protocollo probabilistico che risolve questo problema sotto le precedenti assunzioni (e senza l'assunzione di ID unico). L'idea è abbastanza semplice: ogni nodo sceglie uniformemente a caso e in maniera indipendente un numero nell'insieme $[m]$, il quale diventerà l'ID del nodo. A questo punto basta eseguire uno dei protocolli deterministici conosciuti per la leader election su un anello con l'assunzione di ID unico. Certamente potrebbe capitare che gli identificatori dei nodi non siano univoci, però il valore m è scelto in modo tale che con alta probabilità non ci saranno due nodi x, y tali che $id(x) = id(y)$. Rifacendosi alla definizione di protocollo probabilistico, in questo caso il risultato dell'esecuzione del protocollo è una **variabile aleatoria**. Infatti noi vogliamo che lo stato finale sia uno stato in cui un solo nodo sia nello stato di `leader`, e tutti gli altri nello stato di `follower`. Con questo protocollo probabilistico non abbiamo tale certezza di arrivare ad una configurazione finale accettabile, però sappiamo che arriveremo a tale configurazione desiderata con alta probabilità

7.2.1 Analisi

Riferiamoci al protocollo appena descritto come `random-leader` (`RL`) e alla sua esecuzione $\texttt{RL}(G, n, m)$, dove n è il numero di nodi dell'anello G ed m è il parametro di scelta delle etichette, che vedremo alla fine di questa analisi come scegliere in modo tale che con alta probabilità il protocollo porti ad una configurazione accettabile. Osserviamo che un *evento sufficiente* (ma non *necessario*) affinché il protocollo termini correttamente è il seguente:

$\mathcal{E}$ = "Non esistono due nodi differenti $u, v \in V$, con $u \neq v$, tali che $id(u) = id(v)$"

L'evento $\mathcal{B}$ non è una condizione necessaria in quanto potrebbero esserci due nodi u, v che stessa etichetta, ma se esiste un altro nodo w con etichetta maggiore $id(w) > id(u) = id(v)$, allora tutti i protocolli deterministici pter la leader election già visti porteranno ad una configurazione finale accettabile. Viceversa, se occorre l'evento $\mathcal{E}$, ovvero se tutti i nodi hanno etichette differenti, allora certamente il protocollo porterà ad una configurazione finale accettabile. Osserviamo ora che l'azione di scegliere un'etichetta uniformemente a caso in $[m]$ equivale all'azione di pescare una pallina in un'urna di m palline enumerate da 1 a m. Per calcolare la probabilità dell'evento desiderato $\mathcal{E}$ conviene calcolare la probabilità dell'evento complemetare:

$\overline{\mathcal{E}}$ = "Esistono almeno due nodi differenti $u, v \in V$, con $u \neq v$, tali che $id(u) = id(v)$"

Possiamo calcolare la probabilità di questo evento come l'unione degli eventi *disgiunti*:

$$\overline{\mathcal{E}_i} = \text{"Esistono almeno due nodi differenti con etichetta } i\text{"}$$

Tale evento a sua volta può essere modellato con l'unione di variabili aleatorie binomiali:

$$\mathcal{P}(\overline{\mathcal{E}_i}) = \mathcal{P}(\cup_{k=2}^{n}\{\text{Esistono esattamente } k \text{ nodi differenti con etichetta } i\})$$

Dato che che gli eventi dell'unione sono disgiunti, possiamo scriverla come somma dei singoli eventi:

$$\begin{aligned}
\mathcal{P}(\overline{\mathcal{E}_i}) &= \mathcal{P}(\cup_{k=2}^{n}\{\text{Esistono esattamente } k \text{ nodi differenti con etichetta } i\}) = \\
&= \sum_{k=2}^{n} \mathcal{P}(\{\text{Esistono esattamente } k \text{ nodi differenti con etichetta } i\}) = \\
&= \sum_{k=2}^{n} \binom{n}{k} (\frac{1}{m})^k (1 - \frac{1}{m})^{m-k} \leq \\
&\leq \sum_{k=2}^{n} \binom{n}{k} (\frac{1}{m})^k
\end{aligned}$$

È possibile usare l'approssimazione di Stirling per semplificare il termine della sommatoria:

$$\begin{aligned}
\mathcal{P}(\overline{\mathcal{E}_i}) &\leq \sum_{k=2}^{n} \binom{n}{k} (\frac{1}{m})^k \leq \\
&\leq \sum_{k=2}^{n} (\frac{en}{k} \cdot \frac{1}{m})^k = \\
&= \frac{e^2 n^2}{4m^2} + \sum_{k=3}^{n} (\frac{en}{k} \cdot \frac{1}{m})^k \leq \\
&\leq O(\frac{n^2}{m^2}) + \sum_{k=3}^{n} (\frac{en}{k} \cdot \frac{1}{m})^k = \\
&= O(\frac{n^2}{m^2}) + \sum_{k=3}^{n} O(\frac{n^k}{m^k})
\end{aligned}$$

Dato che $m \geq n$, abbiamo che:

$$\begin{aligned}
\mathcal{P}(\overline{\mathcal{E}_i}) &\leq O(\frac{n^2}{m^2}) + \sum_{k=3}^{n} O(\frac{n^k}{m^k}) \leq \\
&= O(\frac{n^2}{m^2}) + O(\frac{n^3}{m^3}) \leq \\
&= O(\frac{n^2}{m^2} + \frac{n^4}{m^3})
\end{aligned}$$

Dopodiché possiamo calcolare la probabilità $\overline{\mathcal{B}}$ come l'unione di tutte le probabilità $\overline{\mathcal{E}_i}$ per ogni $i \in [m]$:

$$\begin{aligned}\mathcal{P}(\overline{\mathcal{E}_i}) &= \mathcal{P}(\cup_{i=1}^{m} \overline{\mathcal{B}_i}) \leq \\ &\leq \sum_{i=1}^{m} \mathcal{P}(\overline{\mathcal{B}_i}) \leq \\ &\leq m \cdot O(\frac{n^2}{m^2} + \frac{n^4}{m^3}) = \\ &= O(\frac{n^2}{m} + \frac{n^4}{m^2})\end{aligned}$$

Ponendo ora $m = n^3$ avremo che tale probabilità sarà al più:

$$\mathcal{P}(\overline{\mathcal{E}_i}) \leq O(\frac{1}{n} + \frac{1}{n^2}) \in O(\frac{1}{n})$$

Possiamo quindi concludere che per $m \geq n^3$ l'esecuzione del protocollo $\texttt{RL}(G, n, m)$ porterà la rete in una configurazione finale accettabile con alta probabilità, poiché:

$$\mathcal{P}(\mathcal{E}) \geq 1 - O(\frac{1}{n})$$

Capitolo 8

Un protocollo randomizato per reti radio sconosciute

8.1 Un lower bound per i protocolli deterministici

Precedentemente è stato introdotto il protocollo `round-robin` per il task del broadcast a singola sorgente su reti wireless sconosciute con presenza d'interferenza, ed è stato dimostrato che questo protocollo porta a termine il suo task in tempo $O(Dn)$, dove D è il diametro della rete di n nodi considerata. Avvalendosi di altri costrutti matematici e aggiungendo conoscenza ai nodi è possibile ottenere risultati migliori. Per esempio, dato l'insieme $[n]$ e un $k \leq n$ diremo che la famiglia di sottoinsiemi $H := \{H_1, H_2, ..., H_t\}$ è (n,k)**-selettiva** se per ogni sottinsieme $S \subseteq [n]$ con $|S| \leq k$ esiste almeno un $H_i \in H$ tale che $|S \cap H_i| = 1$. Se supponiamo che tutti i nodi conoscano la stessa famiglia H (n,Δ)-selettiva di V, dove $\Delta = \max_{v \in V}\{d(v)\}$, allora è possibile definire un protocollo deterministico che termina il task in questione in tempo $O(D \cdot |H|)$, con $D = diam(G)$. Nell'articolo *"Round Robin is optimal for fault-tolerant broadcasting on wireless networks"* viene dimostrato che per valori sufficientemente grandi di n esiste sempre una famiglia n,k)-selettiva di dimensione $O(k \log n)$. Tale risultato implica che sotto queste assunzioni è possibile definire un protocollo che termina il task in tempo $O(D\Delta \log n)$. Purtroppo però senza queste assunzioni non è possibile fare deterministicamente meglio.

Teorema (8.1): Dato un grafo diretto di trasmissione $G = (V, E)$ relativo ad una rete wireless, con diametro D, grado massimo Δ tale che $D\Delta < n$, allora esiste sempre un modo di manipolare i ritardi in modo tale che non si possa risolvere il problema del broadcasting a singola sorgente in un tempo migliore di $\Omega(D\Delta \log(\frac{n}{D}))$.

Per esempio, se consideriamo un grafo con diametro costante, per esempio $D = 3$, avremo che il grado massimo dei nodi Δ crescerà molto velocemente al crescere di n, per esempio in maniera lineare ($\Delta \in O(n)$). Perciò in questo caso avremo un lowerbound di $\Omega(n \log n)$. Quello che ci si chiede è se esiste un modo per abbattere il fattore Δ del lower-bound. Il teorema precedentemente enunciato ci dice che questo è *deterministicamente impossibile*, perciò è necessario ricorrere alla randomness.

8.2 Protocollo randomizzato per reti radio sconosciute con interferenze

Definiamo un protocollo probabilistico molto semplice, in cui ogni nodo informato trasmette ai suoi vicini con una certa probabilità p fissata e uguale per tutti. È facile mostrare un controesempio il tempo medio di completamente del protocollo è pessimo.

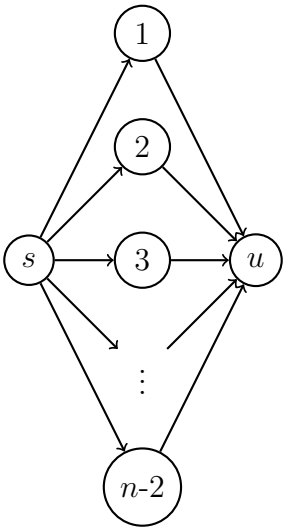


Figura 8.1: Controesempio in cui il tempo medio di completamento è pessimo.

Sia s la sorgente, sicuramente esiste un tempo t_0 in cui s trasmette il messaggio a tutti i suoi vicini $1, 2, 3, ..., n-2$ senza interferenze. Dallo slot temporale successivo tutti i suoi vicini inizieranno a trasmettere verso u, ognuno con probabilità p. Supponiamo per esempio che $p = \frac{1}{2}$, e cerchiamo di stabilire entro quanto tempo ci si aspetta che p verrà informato senza alcuna interferenza. Per prima cosa dobbiamo calcolare con quale probabilità l'evento $\mathcal{E}$ = "v viene informato" occorre, e questo si può fare usando la distribuzione binomiale:

$$\mathcal{P}(\mathcal{E}) = \binom{n-2}{1}\frac{1}{2}(1-\frac{1}{2})^{n-3} = \frac{n-2}{2^{n-2}} \in \Theta(\frac{n}{2^n})$$

Ciò implica che il tempo di completamento del task di questo protocollo nel controesempio in questione è di circa $\Theta(\frac{2^n}{n})$. Anche se v avesse molti meno vicini, per esempio $\sqrt{n}$, il tempo di completamento sarebbe pessimo (ovvero esponenziale, $\Theta(\frac{2^{\sqrt{n}}}{\sqrt{n}})$). Se invece v avesse solamente 2 vicini che trasmettono, il protocollo terminerebbe in tempi brevissimi (sempre considerando questo esempio). Ciò ci suggerirebbe che dobbiamo trovare un valore di p in modo tale che è poco probabile che avvengano interferenze, ovvero un valore per il quale in media trasmette un solo vicino di ogni nodo. COnsiderando sempre il precedente controesempio, il valore ottimale di p per il quale i vicini di v devono trasmettere è $p = \frac{1}{d_{in}(v)}$, ovvero il grado entrante di v.

8.2.1 Protocollo `BGI` su grafi a livelli d-regolari

Come primo passo consideriamo grafi con strutture che in qualche modo ci semplificano il valoro. Consideriamo grafi a livelli d-regolari, ovvero grafi suddivisi in strati

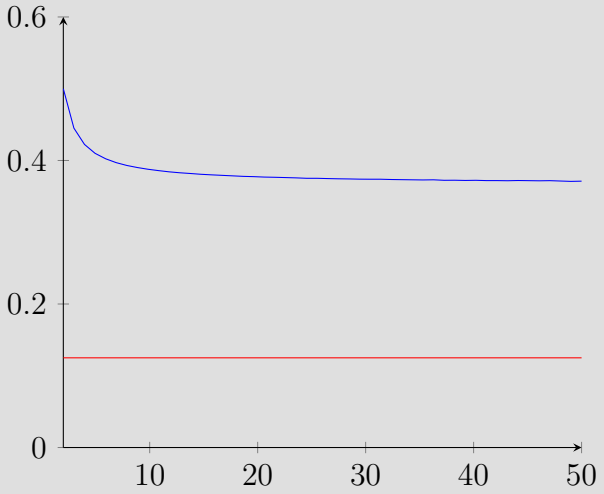
$L_1, L_2, ..., L_D$ in cui esistono archi solamente da un livello a quello successivo, e in cui ogni nodo v ha un grado entrante costante uguale a $d_{in}(v) = d$. Inoltre esiste una sorgente s tale che $\forall v \in L_1, (s, v) \in E$. Osservare che la descrizione appena data non è propriamente precisa, in quanto i nodi in L_1 hanno grado entrante pari ad 1, e quindi non necessariamente d, mentre il grado entrante della sorgente è pari a 0. Definiamo quindi il protocollo `BGI` per tale modello di grafo: per ogni stage $k \geq 1$, per ogni slot temporale $j \in [1, c \log n]$, controllo per ogni nodo x se è stato informato allo stage $k-1$, e nel caso in cui sia stato informato allora trasmetterà a tutti i suoi vicini con probabilità $p = \frac{1}{d}$.

Teorema (8.2): Il protocollo `BGI` su grafi d-regolari e a D strati completa con alta probabilità il task del broadcast dalla sorgente s in $O(D)$ stage e quindi in $O(D \log n)$ slot temporali.

La dimostrazione verrà fatta per induzione sui livelli $j \in [1, D]$, ovvero si vuole dimostrare che al termine dello stage j con alta probabilità tutti i nodi del livello L_j saranno informati. Il caso base per $j = 1$ è banale, perciò assumiamo come ipotesi induttiva che non alta probabilità entro la fine dello stage $j > 1$ tutti i nodi nel livello L_j sono già stati informati (ovvero entro $)(j \log n)$ slot temporali). Si vuole ora dimostrare che questo è vero anche per $j + 1$. Consideriamo quindi un nodo $v \in L_{j+1}$. Iniziamo col calcolare la probabilità che v viene informato in un dato slot temporale della fase $j+1$. Tale probabilità è pari alla probabilità che uno solo dei suoi d vicini entranti trasmette, mentre gli altri $d-1$ "stanno zitti", e ciò avviene con probabilità:

$$\binom{d}{1}\frac{1}{d}(1-\frac{1}{d})^{d-1} = d\frac{1}{d}(1-\frac{1}{d})^{d-1} = (1-\frac{1}{d})^{d-1}$$

Osserviamo che per valori di $d \geq 2$ la quantità $(1-\frac{1}{d})^{d-1} > \frac{1}{8}$.



Non ha senso considerare $d = 1$ in quanto non ci sarebbe la necessità di definire un protocollo probabilistico. Proseguendo, possiamo dire che il nodo v al livello $j+1$ non viene informato in uno slot temporale con probabilità al più $1-\frac{1}{8}$. Perciò la probabilità che v non venga informato in tutti i $c \log n$ slot temporali dello stage $j+1$ è al

più $(1-\frac{1}{8})^{c\log n} < e^{-\frac{c}{n}\log n} = \frac{1}{n^{\frac{c}{8}}}$. Ricapitolando, abbiamo dimostrato che la probabilità dell'evento $\mathcal{E}_v =$ "$v \in L_{j+1}$ non viene informato entro la fine dello stage $j+1$" occorre con al più come $\mathcal{P}(\mathcal{E}_v) \leq \frac{1}{n^{\frac{c}{8}}}$. Per la union bound, possiamo dire che la probabilità che esiste almeno un nodo al livello L_{j+1}, che non viene informato entro la fine dello stage $j+1$, occorre con al più:

$$\mathcal{P}(\cup_{v\in L_{j+1}}\mathcal{E}_v) \leq \sum_{v\in L_{j+1}} \mathcal{P}(\mathcal{E}_v) < n\frac{1}{n^{\frac{c}{8}}} = \frac{1}{n^{\frac{c}{8}-1}}$$

Perciò, ponendo $c = 16$, avremo che la probabilità di avere un cosiddetto *"bad stage"* in cui non tutti i nodi del relativo livello vengono informati è al più $\frac{1}{n}$. Possiamo quindi concludere dicendo che ogni nodo $v \in L_{j+1}$ viene informato entro la fine dello stage $j+1$ con probabilità almeno $1-\frac{1}{n}$, implicando così l'ipotesi del teorema.

Capitolo 9

I modelli gossip

9.1 Le reti opportunistiche - I modelli gossip

I **modelli gossip** lavorano in un contesto *totalmente sincronizzato*, ovvero dove le comunicazioni sono scandite da un *clock globale*. Il ritmo del clock globale è *discreto*, e le singole unità temporali sono chiamata *slot temporali*. Infine si assume che le reti distribuite sulle quali eseguire questi protocolli sono grafi non diretti.

9.1.1 Protocollo `PUSH`

Ad ogni slot temporale $t \geq 0$ ogni nodo u sceglie uniformemente a caso un suo vicino $v \in N(v)$, e se necessario gli trasmette il messaggio m. Entro il termine dello slot temporale t il nodo v riceverà il messaggio m.

9.1.2 Protocollo `PULL`

Ad ogni slot temporale $t \geq 0$ ogni nodo u sceglie uniformemente a caso un suo vicino $v \in N(v)$, e se v ha un'informazione che u desidera allora u si farà trasmettere un messaggio m contenente tale informazione. Entro il termine dello slot temporale t il nodo u riceverà il messaggio m.

9.1.3 Proprietà

Osserviamo che ad ogni istante t tutte le operazioni di `push` o `pull` generano un **grafo dirette delle comunicazioni** $\overrightarrow{G}_t = (V, \overrightarrow{E}_t)$. Cosniderando il protocollo `PUSH`, esiste un arco diretto $(u, v) \in \overrightarrow{E}_t$ se u fa un'operazione di `push` su v. Analogamente per il protocollo `PULL`, esiste un arco diretto $(v, u) \in \overrightarrow{E}_t$ se u fa un'operazione di `pull` su v. Osservare che il grafo $\overrightarrow{G}_t$ risulta *sparso*, ovvero con n archi, perché ci sarà un arco per ogni operazione di `push` o `pull` fatta dai nodi. Perciò possiamop vedere l'uso dei protocolli di tipo `PUSH` o `PULL` come una sequenza di grafi diretti di comunicazione $\{\overrightarrow{G}_t = (V, \overrightarrow{E}_t)\}_{t \geq 0}$.

9.2 Il broadcast su una clique con un protocollo di tipo PULL

Consideriamo il solito problema del broadcast a singola sorgente s su una clique K_n, e definiamo dei protocolli probabilistici di tipo PULL che risolvono il problema del broadcast.

9.2.1 Protocollo banale

Un protocollo banale, che chiameremo `broadcast-pull` (`BP`), è il seguente:

- Ogni nodo, eccetto la sorgente s, inizia il protocollo nello stato `not_informed`.
- Ad ogni tempo $t \geq 0$ ogni nodo $u \neq s$ nello stato `not_informed` fa un'operazione di `pull` su un altro nodo v uniformemente a caso. Se $v \equiv s$, allora u si fa inviare una copia del messaggio m ed entra nello stato `informed`.
- Il protocollo termina globalmente quanto tutti i nodi avranno ricevuto una copia del messaggio m.

Analizziamo ora l'efficienza di tale protocollo. Ciò che vogliamo sapere è in media quanto tempo ci mette un nodo u a fare `pull` sulla sorgente s. Sia la variabile aleatoria $X_u^{(time)}$ che indica entro quanto tempo u diventa informato. Abbiamo che tale variabile aleatoria è una variabile *geometrica* di parametro $p = \frac{1}{n-1}$, ovvero:

$$\begin{aligned}
\mathcal{P}(X_u^{(time)} = 1) &= \mathcal{P}(u \text{ pulls } s) = \frac{1}{n-1} \\
\mathcal{P}(X_u^{(time)} = 2) &= (1 - \mathcal{P}(u \text{ pulls } s)) \cdot \mathcal{P}(u \text{ pulls } s) = (1 - \frac{1}{n-1})\frac{1}{n-1} \\
\mathcal{P}(X_u^{(time)} = 3) &= (1 - \mathcal{P}(u \text{ pulls } s))^2 \cdot \mathcal{P}(u \text{ pulls } s) = (1 - \frac{1}{n-1})^2\frac{1}{n-1} \\
&\vdots \\
\mathcal{P}(X_u^{(time)} = k) &= (1 - \frac{1}{n-1})^k\frac{1}{n-1}
\end{aligned}$$

Con media $\mathbb{E}[X_u^{(time)}] = \frac{1}{p} = n - 1$. Perciò in media dopo $n-1$ slot temporali tutti i nodi verranno informati.

9.2.2 Protocollo migliorato

Un protocollo migliore del precedente, che chiameremo `broadcast-pull+` (`BP+`), è il seguente:

- Inizialmente la sorgente s è nello stato `informed`, mentre il resto dei nodi parte dallo stato `not_informed`.
- Ad ogni tempo $t \geq 0$ ogni nodo u `not_infomed` fa un'operazione di `pull` su un altro nodo v uniformemente a caso. Se v è un nodo `infomed`, allora u si fa inviare una copia del messaggio m ed entra nello stato `informed`.
- Il protcollo termina globalmente quando tutti i nodi entrano nello stato `informed`.

Teorema (9.1): Il protocollo `BP+` termina il suo task sull'istanza ($K_{n,s} \in [n]$) in tempo $\Theta(\log n)$ con alta probabilità.

Teorema (9.1): Fissiamo uno slot temporale $t \geq 0$, e definiamo gli insiemi:

- $I_0 = \{s\}$.
- $I_t = \{v \in V | v \text{ è informato al tempo } t\}$.

La dimostrazione si suddividerà in tre fasi:

- *Fase 1*, in cui si dimostrerà che con alta probabilità il numero di nodi `informed` $|I_t|$ cresce in maniera esponenziale fino a $\frac{n}{2}$.
- *Fase 2*, in cui si dimostrerà che con alta probabilità il numero di nodi restanti `not_informed` decresce in maniera esponenziale.
- *Fase 3*, in cui si dimostrerà che con alta probabilità, una volta rimasti $O(\log n)$ nodi `not_informed`, in un solo slot temporale verranno informati.

Così facendo verrà dimostrato che in tempo logaritmico il protcollo terminerà il suo task con alta probabilità.

- *Fase 1:* Sia $t_{max} = \max\{t \geq 0 | m_t \leq \frac{n}{2}\}$. Fissato un tempo $0 \leq t \leq t_{max}$, definiamo la variabile aleatoria binaria Y_u che indica la probabilità che un nodo `not_informed` diventa `informed` al tempo $t+1$.

$$\forall v \in V \backslash I_t, Y_u = \begin{cases} 1 \text{ se } u \text{ vienme infomato al tempo } t+1 \\ 0 \text{ altrimenti} \end{cases}$$

  Sia $|I_t| = m_t$, allora dato che ogni nodo fa `pull` in maniera uniformemente a caso avremo che

$$\mathcal{P}(Y_u = 1 \,|\, |I_t| = m_t) = \frac{m_t}{n-1} \cong \frac{m_t}{n}$$
$$\mathbb{E}[Y_u \,|\, |I_t| = m_t] = \frac{m_t}{n} \leq \frac{m_{t_{max}}}{n} \leq \frac{1}{2}$$

  Definiamo ora l'insieme I^* l'insieme dei soli nuovi nodi informati al tempo $t+1$, ovvero tale che $I_{t+1} = I_t \cup I^*$. Poiché I_t e I^* sono disgiunti avremo che $|I_{t+1}| = |I_t| + |I^*|$. Osserviamo che $|I^*|$ è una variabile aleatoria composta dalla somma delle variabili aleatorie Y_u per ogni $v \in V \backslash I_t$. Per linearità del valor medio avremo che:

$$\mathbb{E}[|I^*|] = \sum_{v \in V \backslash I_t} \mathbb{E}[Y_u | I_t = m_t] = (n - m_t)\frac{m_t}{n} > \frac{n}{2} \cdot \frac{m_t}{n} = \frac{m_t}{2}$$

  Avremo quindi che l'insieme dei nodi informati cresce come:

$$\mathbb{E}[|I_{t+1}| \,|\, |I_t| = m_t] = m_t + \mathbb{E}[|I^*|] \geq m_t + \frac{m_t}{2} = \frac{3}{2} m_t$$

Abbiamo ottenuto quindi una relazione ricorsiva che dice che il numero di nodi informati al tempo $t+1$ cresce di un fattore costante rispetto al tempo t (finché $m_t \leq \frac{m}{2}$). Perciò "srotolando" l'equazione avremo:

$$m_t \geq \frac{3}{2} m_{t-1} \geq (\frac{3}{2})^2 m_{t-2} \geq ... \geq (\frac{3}{2})^t$$

Sia ora il tempo $\tau = \min\{t \geq 1 | m_t > \frac{n}{2}\}$ il primo istante in cui il numero di nodi `informed` supera la metà dei nodi. Secondo l'equazione di ricorrenza precedente, avremo che $\tau \cong \log_{\frac{3}{2}}(\frac{n}{2}) \in \Theta(\log n)$, ovvero che in tempo logaritmico almeno la metà dei nodi verrà informata. Purtroppo abbiamo solo dimostrato ciò che accade in media, ovvero abbiamo fatto un'analisi *mean field*. Il teorema richiede invece l'alta probabilità. Per dimostrare l'alta probabilità suddividiamo la fase 1 in altre due sottofasi, in cui verrà dimostrato l'alta probabilità della prima fase suddividendo per $1 \leq m_t \leq \alpha \log n$ ed $\alpha \log n < m_t \leq \frac{n}{2}$ (per qualche valore di α).

- *Sottofase 1.1:* Poniamo $1 \leq m_t \leq \alpha \log n$. Per ogni $\alpha > 0$ esiste un $\gamma = \gamma(\alpha)$ sufficientemente grande tale che dopo i primi $\tau_1 = \gamma \log n$ slot temporali, avremo almeno $m_{\tau_1} \geq \alpha \log n$ nodi informati con alta probabilità. Consideriamo per ogni $u \in V \backslash \{s\}$ la variabile aleatoria binaria $Y_u^{(\tau_1)}$ che, come definita prima vale 1 se il nodo u risulta informato al tempo $\tau_1 + 1$, e vale 0 se entro i primi τ_1 passi non viene mai informato. Per indipendenza degli eventi di `pull` avremo che:

$$\begin{aligned}
\mathcal{P}(Y_u^{(\tau_1)} = 0) &= \mathcal{P}(\cap_{i=0}^{\tau_1} u \text{ doesn't pull the message at time } t) = \\
&= \Pi_{i=0}^{(\tau_1)} \mathcal{P}(u \text{ doesn't pull the message at time } t) = \\
&= \Pi_{i=0}^{(\tau_1)} (1 - \frac{|I_t|}{n}) \leq (1 - \frac{1}{n})^{\tau_1} \leq \\
&\leq (e^{-\frac{1}{n}})^{\tau_1} = e^{-\gamma \frac{\log n}{n}}
\end{aligned}$$

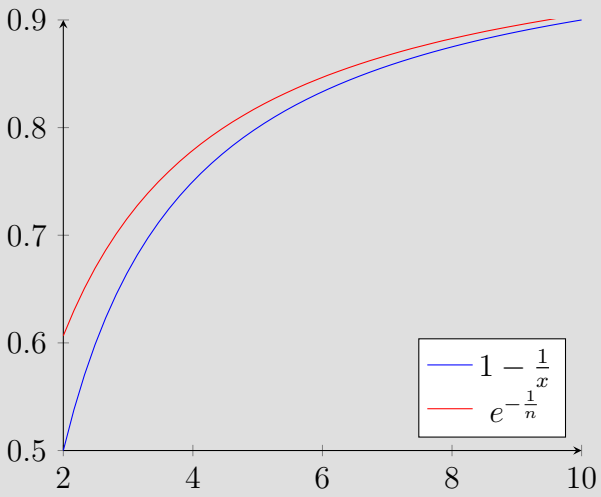


Figura 9.1: $1 - \frac{1}{n} \leq e^{-\frac{1}{n}}$

Sia la variabile aleatoria $Y^{(\tau_1)}$ che indica il numero di nodi che al tempo $\tau_1 + 1$ non risultano ancora informati, composta dalla somma di tutti i $Y|u^{(\tau_1)}$. Osserviamo che in poche parole $Y^{(\tau_1)} = m_{\tau_1}$. In media il suo valore sarà:

$$\mathbb{E}[Y^{(\tau_1)}] = \sum_{u \in V} \mathbb{E}[Y_u^{(\tau_1)}] = \sum_{u \in V} (1 - e^{-\gamma \frac{\log n}{n}}) \geq n \frac{\gamma \log n}{2n} = \frac{\gamma \log n}{2}$$

L'ultima disuguaglianza è data dal fatto che per un n sufficientemente grande vale $1 - e^{-\gamma \frac{\log n}{n}} \geq n \frac{\gamma \log n}{2n}$

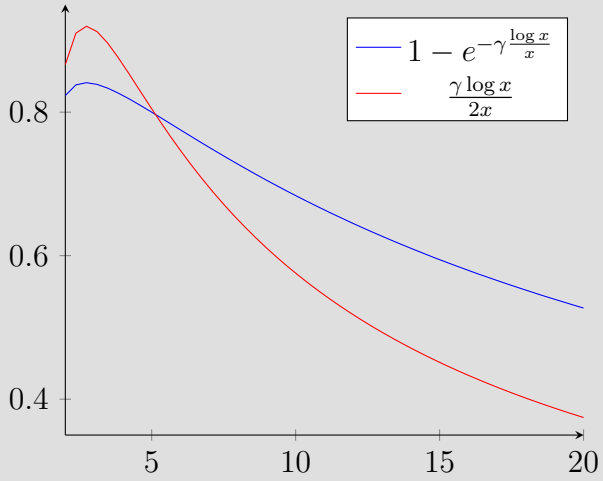


Figura 9.2: Funzioni per $\gamma = 5$

Rimarcando il fatto che $Y^{(\tau_1)}$ è la somma di variabili aleatorie indipendenti tra di loro, è possibile applicare il Chernoff Bound nella sua forma moltiplicativa, ottenendo così che la probabilità di avere meni di $\gamma \frac{\log n}{2}$ entro i primi $\tau_1 = \gamma \log n$ slot temporali è al più:

$$\mathcal{P}(Y^{(\tau_1)} \leq (1 - \frac{1}{2})\gamma \log n) \leq e^{-\frac{\gamma \log n}{8}} = n^{-\frac{\gamma}{8}}$$

Ponendo un $\gamma = \gamma(\alpha)$ sufficientemente grande tale che $\frac{\gamma(\alpha) \log n}{2} \geq \alpha \log n$, otterremo che $m_{\tau_1} \geq \alpha \log n$ con alta probabilità.

$$\mathcal{P}(Y^{(\tau_1)} \leq \alpha \log n) \leq \mathcal{P}(Y^{(\tau_1)} \gamma(\alpha) \frac{\log n}{2}) \leq n^{-\frac{\gamma(\alpha)}{8}} \Rightarrow$$
$$\Rightarrow \mathcal{P}(Y^{(\tau_1)} \geq \alpha \log n) \geq 1 - n^{-\frac{\gamma(\alpha)}{8}}$$

– *Sottofase 1.2*: Poniamo $\alpha \log n < m_t \leq \frac{n}{2}$. Essite un certo β sufficientemente grande tale che dopo $\tau_2 = \beta \log n$ slot temporali avremo che $m_{\tau_2} \geq \frac{n}{2}$ con alta probabilità. Fissato un $\tau_1 \leq t \leq \tau_2$, sia $m_t = |I_t|$. Notiamo che m_t non è una variabile aleatoria in quanto stiamo considerando un t fissato. Sia quindi la variabile aleatoria binaria Y_u^t che vale

1 se il nodo $u \in V \backslash I_t$ risulta informato al tempo $t+1$ o 0 altrimenti. Grazie alla prima parte della dimostrazione sappiamo che:

$$\mathbb{E}[|I_{t+1}| \, | \, |I_t| = m_t] = \mathbb{E}[\sum_{u \in V} Y_u^t] \geq \frac{3}{2} m_t$$

E siccome stiamo considerando un $t \geq \tau_1$, dalla dimostrazione sopra abbiamo che:

$$\mathbb{E}[m_{t+1}] = \mathbb{E}[|I_{t+1}| \, | \, |I_t| = m_t] \geq \frac{3}{2} m_t \geq \alpha \log n$$

Importante osservare che m_{t+1} è una variabile aleatoria che dipende da m_t, il quale ribadiamo essere un valore fissato. Dato che m_{t+1} è la somma delle variabili aleatorie indipendenti Y_u^t, possiamo nuovamenten applicare il Chernoff Bound e ottenere:

$$\mathcal{P}(m_{t+1} \leq (1-\delta)\frac{3}{2} m_t) \leq e^{-\frac{\delta^2}{2}\frac{3}{2} m_t} \leq e^{-\frac{\delta^2}{2}\alpha \log n} = n^{-\frac{\delta^2}{2}\alpha}$$

Ponendo $\delta \geq \frac{1}{3}$ avremo che $\mathcal{P}(m_{t+1} \leq m_t) \leq n^{-\frac{\delta^2}{2}\alpha}$. Quest'ultima limitazione implica l'alta probilabilità della crescita esponenziale di m_t, ovvero $\mathcal{P}(m_{t+1} > m_t) \geq 1 - n^{-\frac{\delta^2}{2}\alpha}$.

- *Fase 2:* Questa fase è del tutto analoga alla precedente, con la differenza che basta dimostrare che la decrescita dei nodi non informati è esponenziale nel tempo. Fissato un tempo $t \leq \tau_2 = \beta \log n$. Dato che dalla prima fase sappiamo che $m_{\tau_2} \geq \frac{n}{2}$, avremo che un nodo `not_informed` al tempo t fa `pull` del messaggio e diventa `informed` al tempo $t+1$ con probabilità almeno $\frac{1}{2}$. Fissiamo con $z_t = n - m_t \geq \frac{n}{2}$ il numero di nodi sopravvissuti al tempo t. Mediamente avremo che il numero di nodi "sopravvissuti" al tempo $t+1$ (ovvero quelli ancora `not_informed`) saranno:

$$z_{t+1} \leq \frac{z_t}{2} \leq \frac{z_\tau}{2} \leq \frac{n}{4}$$

Questa catena si può generalizzare come segue:

$$z_t \leq \frac{z_{t-1}}{2} \leq \frac{z_{t-2}}{4} \leq ... \leq \frac{z_{t-1}}{2^i} \leq ... \leq \frac{z_{tau_2}}{2^{t-\tau_2}} \leq \frac{n}{2^{t-\tau_2+1}}$$

Ci si chiede ora per quali valori di t il numero di nodi sopravvissuti diventa molto piccolo, diciamo $z_t \leq c \log n$. Certamente se $\frac{n}{2^{t-\tau_2+1}} \leq c \log n$ allora anche $z_t \leq c \log n$, allora basta risolvere la prima disuguaglianza per ottenere

la seconda:

$$\begin{aligned}
\frac{n}{2^{t-\tau_2+1}} &= c\log n \\
2^{t-\tau_2} &= \frac{n}{2c\log n} \\
2^t 2^{-\tau_2} &= \frac{n}{2c\log n} \\
2^t 2^{-\beta\log n} &= \frac{n}{2c\log n} \\
2^t n^{-\beta'} &= \frac{n}{2c\log n} \\
2^t &= \frac{n^{\beta'+1}}{2c\log n} \\
t &= \log(\frac{n^{\beta'+1}}{2c\log n}) \\
&= \log(n^{\beta'+1}) - \log(2c\log n) \\
&= (\beta'+1)\log n - \Theta(\log(\log n)) \in \Theta(\log n)
\end{aligned}$$

- *Fase 3:* Calcoliamo infine la probabilità che una volta rimasti $O(\log n)$ nodi `not_informed`, in un solo slot temporale tutti diventano informati, e voediamo che ciò accade con alta probabilità. Per comodità diciamo che sono rimasti $c\log n$ nodi `not_informed`, oer qualche costante $c > 0$. La probabilità che un nodo `not_informed` faccia `pull` su un nodo `informed` in questa fase del protocollo è dell'ordine di $1 - \frac{\log n}{n}$, perciò avremo che in media il numero di nodi che non faranno `pull` su nodi informati sarà:

  $$\mathbb{E}[z_{t+1}|z_t = c\log n] = c\frac{\log^2 n}{n}$$

  Applicando la disuguaglianza di Markov, avremo che la probabilità di avere almeno un sopravvissuto sarà molto bassa:

  $$\begin{aligned}
  &\mathcal{P}(z_{t+1} \geq 1) \leq c\frac{\log^2 n}{n} \Rightarrow \\
  &\Rightarrow \mathcal{P}(z_{t+1} = 0) \geq 1 - c\frac{\log^2 n}{n}
  \end{aligned}$$

Capitolo 10

Gli α-espansori

Generalmente si fanno studi epidemici su *modelli omogenei*, ovvero modelli in cui ogni individuo può entrare in contatto con un altro in maniera *omogenea*, ovvero *uniformemente a caso*. Questi modelli in realtà non sono per niente realistici, in quanto il grafo delle relazioni è una clique. In un contesto reale invece, il grafo delle relazioni che ci sono tra gli individui è molto più sparso di una clique. Perciò quello che ci si può chiedere è se esistono modelli di grafi più sparsi che però non impediscono le diffuzioni. In termini più precisi se esistono modelli di grafo con densità $o(n^2)$ nei quali però il broadcasting di un'informazione o la diffuzione di un'infezione, tramite protocolli di gossip, termina in tempo $\Theta(\texttt{POLY}(\log n))$. Una caratteristica necessario che un grafo di questo tipo dovrebe avere è un *perimetro basso*, perché in un grafo con perimetro massimo (ovvero $n-1$) non c'è modo di diffondere un'informazione in tempo poli-logaritmico. Ancora, dovrebbe avere una buona connettività tra i nodi, ovvero una buona *fault-tollerance*, Descriviamo quindi una famiglia di grafi che hanno tali caratteristiche:

- Dato un grafo $G = (V, E)$ Δ-regolare, ovvero co grado massimo Δ, per ogni sottoinsieme $S \subset V$ la sua **node-expansion** è definita come

$$|N(S)| \text{ dove } N(S) = \{v \in V - S : (u, v) \in E \wedge u \in S\}$$

- Per ogni costante fissata $\alpha \geq 0$, diremo che un grafo G è un **α-expander** se ogni sottoinsieme $S \subset V$ tale che $|S| \leq \frac{1}{2}$, ha espansione almeno $|N(S)| \geq \max\{n, \alpha|S|\}$.

L'interesse nei $\Omega(1)$-expanders è motivato nel seguente teorema:

Teorema (10.1): Consideriamo la famiglia infinita di grafi al crescere della loro dimensione $\{G_n = (V_n, E_n) : V = |n| \wedge n \geq 1\}$. Se esiste una costante assoluta $\alpha > 0$ tale che, per n sufficientemente grande, ogni grafo G_n è un α-expander, allora il diametro di G_n è $O(\log n)$. Inoltre, sotto le stesse assunzioni, per disconnettere totalmente ogni sottoinsieme $S \subset V_n$, con $|S| \leq \frac{1}{2}$, bisogna rimuovere un numero lineare in $|S|$ di archi.

Teorema (10.1): Per dimostrare il primo enunciato verrà fatta una visita `BFS` in ampiezza. Consideriamo una qualsiasi sorgente $s \in V$, ed applichiamo la visita in ampiezza su s che però si ferma non appena visita più di $\frac{1}{2}$. Più precisamente, sia

L_t l'insieme dei nodi al livello $t \geq 0$ dell'albero `BFS` risultante, e sia $I_t = \cup_{i=0}^{t} L_i$ l'insieme di tutti i nodi che appartengono ad un livello al più t, la visita `BFS` si fermerà quando si arriverà ad un livello L_τ, tale che $\tau = \min\{t \leq 1 : |I_t| > \frac{n}{2}\}$.

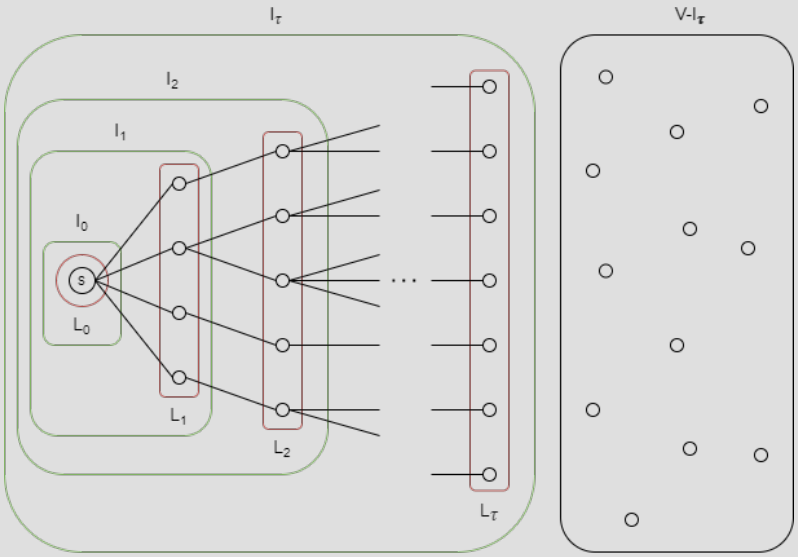


Figura 10.1: Risultato della visita `BFS` a metà.

Quello che ci interessa dimostrare è che I_t cresce in maniera esponenziale. Per prima cosa vediamo come cresce L_t in funzione di I_{t-1}. Per $t = 1$ avremo che $L_1 = N(S) \Rightarrow |L - 1| \geq 1$. Osservando che $L_t = N(I_{t-1})$, possiamo definire la relazione più generale:

$$|I_t| = |I_{t-1}| + |L_t| = |I_{t-1}| + |N(I_{t-1})| \geq |I_{t-1}| + \alpha |I_{t-1} = (1+\alpha)|I_{t-1}||$$

"Srotolando" questa relazione più generale:

$$|I_t| \geq (1+\alpha)|I_{t-1}| \geq (1+\alpha)^2 |I_{t-2}| \geq \ldots \geq (1+\alpha)^{t-1} |I_1| \geq (1+\alpha)^{t-1}$$

Infine ponendo $\tau \cong \log_{(1+\alpha)}(\frac{n}{2}) \in \Theta(\log n)$, otterremo che $|I_\tau| > \frac{n}{2}$. Ciò implica che il numero di nodi a distanza τ da s sono almeno $\frac{n}{2}$. Consideriamo ora un altro nodo $s' \in V - I_\tau$, e ripetiamo lo stesso tipo di visita `BFS` partendo da s'. Grazie ancora la fatto che G è un α-expander, avremo che dopo $\tau' \in \Theta(\log n)$ l'albero `BFS` ha raggiunto $\frac{n}{2}$ nodi con distanze logaritmiche. Dato che entrambe le visite, quella che parte da s e quella che parte da s', raggiungono almeno la metà dei nodi, avremo che i due alberi `BFS` si intersecano. Perciò avremo che s e s' sono distanti al più $\tau + \tau' \in \Theta(\log n)$, implicando così che il grafo G α-expander ha diametro *logaritmico* (per un n sufficientemente grande).

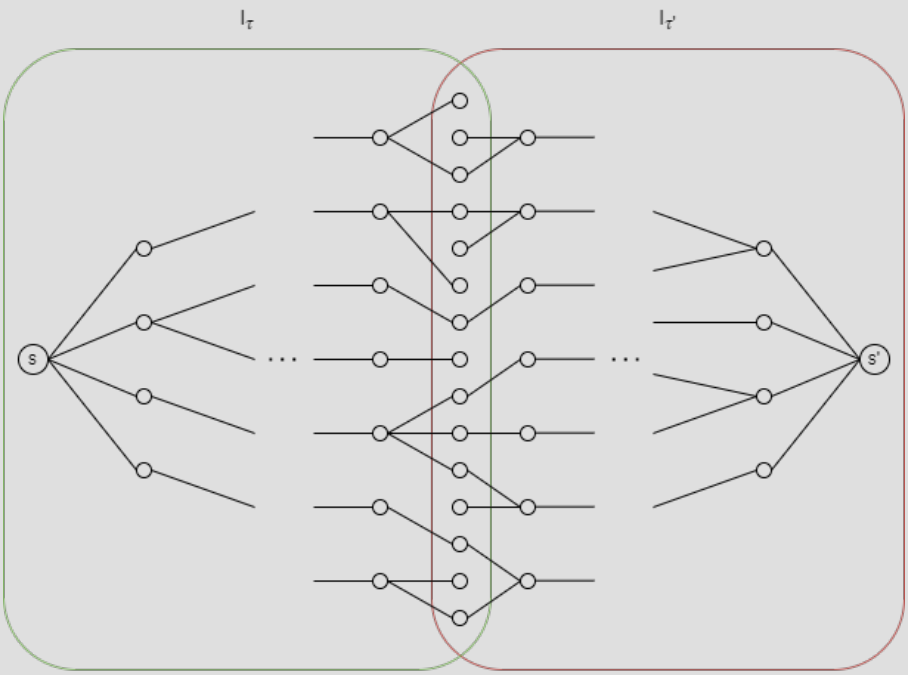


Figura 10.2: Le due visite `BFS` che si incrociano.

Infine per dimostrare la robustezza di G basta osservare che, dato che ogni S in questione ha un'α-espansione, per disconnettere S bisogna rimuovere $\Omega(\alpha|S|)$ archi.

Capitolo 11

Il consenso a maggioranza

Capitolo 12

La colorazione di un grafo distribuita

Parte III

Teoria dei giochi

Capitolo 13

Introduzione