

# Lezione 1 ADRC

## Introduzione generale al corso

Prima parte del corso prof. Clementi

Seconda parte dott. Pepè

**Limiti SD** (Sistema Distribuito)

Esempio

### (*Sistema Standard*)

Consideriamo un grafo  $G(V, E)$ .

Il problema è dato il grafo trovare il MST  $T \subseteq E$

Il grafo è rappresentato con matrice di adiacenza

Nei sistemi standard il problema è semplice dato che si ha una **visione globale** dell'input, ma la sua esecuzione è strettamente sequenziale. Si lavora con un solo processore.

### (*Sistema Distribuito*)

In questo caso, il grafo fa sia da input del problema che da topologia di rete sui cui gira il sistema distribuito

Ho un processore su ogni nodo, che rappresenta una macchina.

A tempo 0 ogni nodo ha una conoscenza limitata dell'input, infatti ogni nodo conosce solo il suo  $ID \in [n]$  e l'etichetta dell'arco con i suoi vicini.

Si ha una **visione locale** dell'input, ma ci sono più processori su cui lavorare

L'algoritmo ha forti limiti sulla comunicazioni tra processori.

Nei SD bisogna in qualche modo determinare cos'è l'output dell'algoritmo, infatti definiamo due tipi di output :

- **output globale** : Ogni nodo, al termine dell'esecuzione dell'algoritmo, deve conoscere la situazione finale (es. nel problema del MST, ogni nodo deve conoscere gli archi che compongono il MST)
- **output locale** : Ogni nodo conosce solamente la parte di soluzione interessata a lui (es. prob. del MST, ogni nodo conosce gli archi del MST incidenti a lui)

Nei SD, l'algoritmo diventa un **insieme di regole**, immerse nel mondo distribuito :

1. Ogni regola va assegnata ad ogni nodo
2. Ogni nodo vede la sua storia e lo stato in cui si trova, poi prende decisioni (esegue passo sotto)
  1. Cambia stato, aggiorna variabili locali secondo le regole, poi decide se e dove mandare messaggi (**azione atomica** al tempo  $t$ )

Bisogna chiarire la differenza tra algoritmo e protocollo nei SD :

- Algoritmo distribuito : **Idea algoritmica**
- Protocollo : **Implementazione dell'idea algoritmica su uno specifico SD**

**Per costruire** il protocollo ci immergiamo in un nodo locale : regole locali sui nodi

**Per analizzare** il protocollo ci mettiamo un'visione globale del sistema

Nei sistemi centralizzati il concetto di spazio e tempo è ben definito

Nei sistemi distribuiti invece dobbiamo considerare non solo il tempo di esecuzione dei processori, ma anche i ritardi di trasmissione dei messaggi.

Prendiamo  $\tau = t \max$  di ritardo di trasmissione

I SD si dividono in due classi :

- **modelli sincroni** :
  - il protocollo conosce  $\tau$ , e questo  $\tau$  può essere codificato
  - il tempo di esecuzione è definibile
- **modelli asincroni** :
  - concetto di tempo non troppo fondamentale
  - il protocollo *non* conosce  $\tau$ 
    - **total reliability** : Ci garantisce che prima o poi il sistema terminerà, non si sa quando ma sempre in un tempo finito.

Nei modelli sincroni troviamo il concetto di **tempo parallelo**, ovvero il numero di round affinché tutti i processori raggiungano lo stato di terminazione.

# Lezione 2 ADRC - Struttura del modello, Assiomi e Restrizioni

## Struttura del modello

Per le definizioni formali ritornare con gli appunti di Clementi

Definiamo cos'è un Sistema Distribuito

### Sistema Distribuito >

Un SD è un **insieme di entità** (dette anche **nodi, agenti, etc..**) che eseguono computazioni locali e possono comunicare fra loro, mediante link di comunicazione. Ogni nodo nel SD ha un **clock locale**, che scandisce il tempo per le computazioni locali del nodo.

I vari clock locali dei nodi non per forza sono sincronizzati fra loro

Il SD si rappresenta con una struttura a grafo  $G$ , dove :

- $V$  è l'insieme dei nodi/entità
- $E$  è l'insieme dei link di comunicazione (archi)

La comunicazione fra le entità è dettata dallo scambio di messaggi, vedremo più avanti come sono fatti

## Operazioni e stato del nodo

Le operazioni che un singolo nodo può effettuare sono molteplici, tra cui :

- Elaborare e salvare informazioni in locale (computazioni locali)
- Trasmettere messaggi ad altre entità (comunicazione)
- (Re)impostare il proprio clock
- etc..

Ogni nodo ha uno **stato**, che ci va ad indicare lo stato attuale del nodo, ad es. "sleeping", "idle", "computing", etc..

***Lo stato di "idle" è considerato lo stato di default.***

L'insieme degli stati del nodo è definito in questo modo

$$\text{state}(x) = \text{insieme finito di possibili stati in cui il nodo si può trovare}$$

### Attenzione >

È importante ricordare che ad ogni istante di tempo, un nodo può trovarsi in uno e un solo stato ben definito (non abbiamo quindi stati intermedi, ambigui)

## Eventi possibili sul SD, Comportamento di un nodo e del sistema

Ogni entità al tempo 0 si trova nello stato di "idle", cioè dormiente (tranne almeno una chiamata *INITIATOR*, vedi avanti)

Le varie entità possono essere stimolate da eventi esterni al SD, tra cui :

- Tick del clock
- Impulso spontaneo (ad es. impulso generato da un sensore, in una rete di sensori)
- Ricezione del messaggio
- etc..

La **regola** generale che vale per ogni entità è la seguente :

$$\text{stato} \times \text{evento} \rightarrow \text{azione}$$

Ciò significa che un'azione che l'entità esegue è ben definita e determinata dalla coppia (stato,evento)

L'azione di un nodo ha le seguenti proprietà :

- è **atomica** : ciò significa indivisibile, non può essere interrotta
- è **terminale** : ciò significa che l'azione deve terminare entro un tempo finito

Da qui determiniamo il **comportamento** di un'entità

$$B(x) = \text{insieme di regole } \forall \text{ possibile stato ed evento}$$

Il comportamento di un nodo si dice :

- **deterministico** : se alla coppia (stato,evento) è associata **una** sola azione
- **completo** : se  $\forall$  coppia (stato,evento)  $\exists$  un'azione.

### Comportamento del sistema

Possiamo ora definire il comportamento dell'intero sistema, come

$$B = \{B(x) : x \in V\}$$

Tale comportamento si dice **simmetrico** se tutte le entità hanno stesso comportamento

$$B(x) = B(y), \forall x, y \in V$$

# Comunicazioni

Abbiamo detto prima che i nodi possono comunicare fra loro mediante link di comunicazione, scambiandosi messaggi.

Un **messaggio** nel SD altro non è che una sequenza finita di **bit**

Modelliamo il SD come un grafo diretto, usando il *modello point-to-point*

Indichiamo con :

$$\begin{aligned} N_o(x) &= \text{insieme dei vicini uscenti da } x \\ N_i(x) &= \text{insieme dei vicini entranti in } x \\ \forall x \in V \end{aligned}$$

Un qualunque nodo  $x$  può mandare messaggi solo a  $N_o(x)$

Un qualunque nodo  $x$  può ricevere messaggi solo da  $N_i(x)$

# Assiomi/Restrizioni

Definiamo ora un pò di **assiomi/restrizioni** che applicheremo al nostro SD

## Scambio messaggi

1. Il tempo di invio dei messaggi è un tempo finito
2. Ogni entità distingue i suoi vicini (abbiamo porte logiche numerate in locale sul nodo  $x, \forall x$ )

Possiamo avere che per un certo nodo  $a$ , la porta uscente verso il nodo  $b$  sia etichettata con "2", mentre viceversa la porta sia etichettata con "16"

Il protocollo **non si deve** basare su questo tipo di etichettamento

La **topologia** : Grafo etichettato  $(G, \lambda)$

- Per ogni nodo  $x, \lambda_x(x, y)$  indica la numerazione della porta sull'arco da  $x \rightarrow y$

La distinzione dei messaggi vale sia in uscita che in entrata

## Ordinamento messaggi

I messaggi arrivano uno alla volta.

Se arrivano più messaggi su un nodo, essi vengono organizzati secondo il modello **FIFO**:

- il primo arrivato viene letto
- gli altri si accodano in attesa.

Abbiamo quindi una struttura dati ***queue***

Non può accadere che venga letto prima l'ultimo messaggio arrivato

## Link bidirezionali

Se ci troviamo nella situazione in cui

$$\begin{aligned}\forall x, N_i(x) &= N_o(x) = N(x) \wedge \\ \forall y \in N(x) : \lambda_x(x, y) &= \lambda_y(x, y)\end{aligned}$$

Allora possiamo parlare di ***link bidirezionali***

Possiamo quindi trattare gli archi orientati, entranti e uscenti, come archi non orientati, mantenendo l'etichettatura delle porte.

In questo caso, i nodi possono riconoscere chi gli ha inviato il messaggio

## Assunzioni su reliability

- **Guaranteed Delivery** : Ogni messaggi inviato sarà ricevuto senza corruzioni
- **Partial Reliability** : Non ci saranno faults (\*)
- **Total Reliability** : Non ci sono stati faults (nel passato), non ci sono faults e non ci saranno faults a prescindere (in futuro)
  - Quasi sempre useremo questa assunzione

(\*) Faults : Errore che potrebbe capitare nel SD. Si dividono in due tipi :

- Fault-Arco : Messaggio non arriva mai
- Fault-Nodo : Messaggio corrotto

## Restrizioni topologiche

- Grafo fortemente connesso
- Grafo ad anello
- etc..

## Restrizioni su conoscenza

- Conoscenza del numero di nodi
- Conoscenza sul numero di link
- Conoscenza del diametro della rete
- etc..

## Restrizioni sul tempo

- **Bounded Communication Delay** :  $\exists$  costante  $\Delta$  tale che, in assenza di errori, il delay dei messaggi è al più  $\Delta$
- **Clock sincronizzati** : Tutti i clock sono incrementati di una unità in modo simultaneo e ad intervalli costanti

## Misure di complessità - Performance

Abbiamo sostanzialmente due misure per misurare la complessità del protocollo :

1. P.O.V. del sistema : La misura è l'**ammontare di comunicazioni** = numero di bit scambiati da inizio a fine
2. P.O.V dell'utente : La misura è il **tempo**
  1. In questo caso, i vari delay dei messaggi non sono predictabili (non mi ricordo in italiano come si scrive)
  2. **Il tempo ideale** sarebbe : 1 unità di tempo  $\times$  1 messaggio

## Esempio Broadcast

Assunzioni = Restrizioni

- Abbiamo un unico INITIATOR (per def. del problema)
- Ci mettiamo nel caso di Total Reliability con Link Bidirezionali (assunzioni per semplificare)
- $G$  è connesso (altrimenti irrisolvibile)

## Algoritmo FLOOD

**Idea** : Se un'entità sa qualcosa, la manda ai suoi vicini

Abbiamo due tipi di entità :

1. La prima, detta INITIATOR, che è colui che si "sveglia" per primo e fa partire tutto il protocollo.
  1. L'initiator è una sola entità, per assunzione del problema
2. La seconda, detta SLEEPING, che aspetta che un evento esterno la svegli

Le azioni che le due entità eseguono sono le seguenti :

```
INITATOR
Spontaneamente
    invia (I) a N(x)
```

```
SLEEPING
Riceve(I)
```

```
invia (I) a N(x)
```

Fatto in questo modo, il protocollo :

- è corretto
- in tempo finito termina il task (\*)
- **ma** non termina mai (!!!)

Dimostriamo ora (\*) usando il seguente lemma

**Lemma :**

$\forall l \geq 0 \exists$  un tempo  $t \in \mathbb{N}$  tale che

$$\forall v \in L_l(s), v \text{ è "informed"}$$

Definiamo con  $L_l(s)$  i nodi a distanza  $l$  dal nodo  $s$

**dimostrazione per induzione su  $l$**

- Caso Base :  $l = 0 \rightarrow$  trivial
- Caso Induttivo :  $\forall v \in L_{l-1}(s) \rightarrow v$  è "informed"
  - Per il C.I tutti i nodi presenti in  $L_{l-1}(s)$  hanno una copia del messaggio
  - $v \in L_l(s)$  implica che  $\exists w \in L_{l-1} : (w, v) \in E$
  - Per le restrizioni che abbiamo imposto :  $\exists t \in \mathbb{N} : v$  è "informed"

## Ottimizzazione

Come abbiamo visto, il protocollo così fatto risulta essere corretto, e ci garantisce che in un certo tempo finito completa il task.

Il problema sta nel fatto che questo protocollo non termina mai, anche dopo aver completato il task

Vediamo come migliorarlo :

**Idea** : mando a tutti il messaggio tranne che al sender, poi mi metto in un nuovo stato chiamato DONE

- Ogni nodo quindi non ha più due stati {INITIATOR,SLEEPING} ma avrà tre stati {INITIATOR,SLEEPING,DONE}

L'algoritmo modificato è quindi il seguente :

```
IF INITIATOR
  Spontaneamente
    invia (I) a N(x)
    diventa DONE
```

IF SLEEPING

Riceve (I)

    invia (I) a N(x) - {sender}

    diventa DONE

IF DONE

do-nothing

# Lezione 3 ADRC - Time Complexity, LowerBound per il Broadcast, Labeled Hypercube

## Time complexity : tempo ideale

Per calcolare la **time complexity** dei protocolli ci dobbiamo mettere nell'assunzione del modello sincrono

In questa assunzione tutti i ritardi li impostiamo ad "1".

Sotto questa ipotesi, modifichiamo il lemma della lezione scorsa così

### 💡 Lemma\* >

$\forall s \in V$  sorgente,  $\forall h \geq 1, \forall v \in L_h(s)$  abbiamo che :

$v \rightarrow \text{DONE}$  entro  $h$  steps

Nel sistema sincrono, se prendiamo  $v \in L_h, u \in L_{h+1}$  abbiamo che  $v$  viene informato per forza **prima** di  $u$

Di conseguenza,  $\forall v \in V$  si ha che  $d(s, v) \leq n - 1$ , per essere più precisi :

$$\max_x \{d(x, s)\} = \text{eccentricità di } s \leq \text{Diag}(G) \leq n - 1 \implies O(n)$$

Quindi il protocollo termina entro  $n - 1$  steps

Rivedi al volo parte prima di lower bound

## Lower Bound per Broadcast

Ritorniamo al problema Broadcast affrontato in precedenza.

### 💡 Teorem

Sotto le assunzioni

$R = \{\text{UniqueInitiator}(UI), \text{Connectivity}(CN), \text{TotalReliability}(TR), \text{BidirectionalLink}(BL)\}$

ogni protocollo per Broadcast, nel caso peggiore, richiede l'invio di  $m$  messaggi  
 $\Rightarrow \Omega(m)$  **Lower-Bound per la message complexity**

### dim x assurdo

Sia  $A$  un algoritmo che scambia meno di  $m(G)$  messaggio (con  $m(G)$  indichiamo il numero di archi in  $G$ ), allora c'è **almeno** un arco in  $G$  dove non è passata neanche una copia del messaggio

Sia  $e = (x, y)$  tale arco. Prendiamo il grafo  $G$  e da lui creiamo un nuovo grafo  $G'$ , dove aggiungiamo un nuovo nodo  $z$ , lo collegiamo ad  $x, y \rightarrow (z, x) \in E', (z, y) \in E'$  ed eliminiamo l'arco  $(x, y) \in E$

(Questa modifica avviene proprio sul SD, non solo sul grafo).

Di conseguenza :

$$G' = \{V \cup \{z\}, E - e \cup \{(z, x), (z, y)\}\}$$

Ora ripetiamo l'algoritmo  $A$  sia su  $G$  che su  $G'$ , con la stessa sequenza di ritardi.

In  $G$ , i nodi  $x, y$  non si sono mai scambiati il messaggio, questa cosa succederà anche su  $G'$ , e di conseguenza il nodo  $z$  rimarrà nello stato SLEEPING

Così facendo, vediamo che la **tesi** non è valida, quindi  $\Omega(m)$  per forza ■

## Labeled Hypercube

Spostiamoci sull'architettura **Labeled Hypercube**  $\mathcal{H}$

Gli Hypercube sono un tipo di architettura parallela che offre un perfetto compromesso tra  $\deg(G)$  e  $Diam(G)$

Ogni arco di  $\mathcal{H}$  è etichettato dalla dimensione del bit per il quale il nome dei nodi differisce

Negli Hypercube si ha che

$$Diam(\mathcal{H}) = \deg(\mathcal{H}) = \Theta(\log(n))$$

Negli Hypercube di dimensione  $d$  abbiamo che  $|V| = 2^d$

Per costruire un Hypercube di dimensione  $d + 1$  procediamo in questo modo :

1. prendiamo 2 copie di  $\mathcal{H}$  di dimensione  $d$  e le mettiamo parallele (una sopra l'altra)
2. L'etichetta dei nodi seguirà la seguente distribuzione :
  1. Nella copia di sopra aggiungiamo il bit più significativo a 0
  2. Nella copia di sotto aggiungiamo il bit più significativo a 1

3. Otteniamo quindi un quadrato
3. Aggiungiamo quindi le etichette sugli archi
  1. L'etichetta è il bit che cambia quando passo da un nodo ad un altro. Ad es.

$$00 \rightarrow 10$$

cambia il primo bit da dx, quindi l'arco  $(00, 01)$  avrà etichetta 1

2.  $00 \rightarrow 10$  cambia il secondo bit da dx, quindi  $(00, 10)$  avrà etichetta 2

3. e così via

# Lezione 4 ADRC - Algoritmo HyperFLOOD, WakeUp Task, Protocollo WFLOOD

## Ancora su Hypercube

### Ξ Hypercube scalabile >

L'architettura Hypercube si dice **scalabile** se vale che

$$\deg(\mathcal{H}) = \text{Diam}(\mathcal{H}) = \text{poly-log}$$

Ricordiamo che in  $\mathcal{H}$  abbiamo che :

- $|V| = n = 2^d$
- $|E| = \frac{n \log(n)}{2} = \frac{2^d d}{2}$

Definiamo ora la distanza tra due nodi  $\hat{x}, \hat{y} \in \mathcal{H}$ , usando la metrica **distanza di Hamming**

### Ξ Distanza di Hamming >

Siano  $\hat{x}, \hat{y} \in \{0, 1\}^d$  due stringhe di bit (ovvero due nodi in  $\mathcal{H}$ )

Allora, definiamo la distanza di Hamming tra questi due nodi come

$$d_H(\hat{x}, \hat{y}) = |\{i \in [d] : x_i \neq y_i\}|$$

A questo punto possiamo definire il vicinato di un nodo  $\hat{x}$  negli Hypercube

### Ξ Vicinato di un nodo >

Sia  $\hat{x} \in \{0, 1\}^d$  un nodo in  $\mathcal{H}$ .

Il suo vicinato è

$$N(\hat{x}) = \{\hat{y} \in \{0, 1\}^d : d_H(\hat{x}, \hat{y}) = 1\}$$

Vediamo ora qualche proprietà strutturale interessante

$\forall x \in V$

1. **Fatto 1**  $|N(x)| = d = \log_2 n$

2. **Fatto 2**  $\text{diam}(\mathcal{H}_d) = d = \log_2 n$

**dimostrazione fatto 2** (vedi bene slides di Andrea)

$\forall x, y \in V \implies \exists x \rightarrow y$  di dimensione  $\leq d$

Prendiamo i nodi  $x, y$  come sequenze di bit

$$\hat{x} = \langle x_d, x_{d-1}, \dots, x_j, \dots, x_1 \rangle, \hat{y} = \langle y_d, y_{d-1}, \dots, y_j, \dots, y_1 \rangle$$

Da sinistra verso destra :

- Controllo i due bit
  - Se sono uguali non facciamo nulla
  - Altrimenti, prendo il bit di  $\hat{x}$  e riscrivo  $\hat{x}$  con quel bit al complemento. In questo modo avremo che la  $d_H$  sarà pari a 1, e quindi  $\exists$  arco (ho costruito un pezzo del cammino)
  - Vado avanti al più  $d$  volte, e quindi la lunghezza del cammino sarà  $\leq d$  ■

Ora, essendo che  $|E| = \frac{2^d d}{2}$ , l'algoritmo FLOOD avrà msg\_complexity pari a  $\Theta(n \log(n))$

## Algoritmo HyperFLOOD

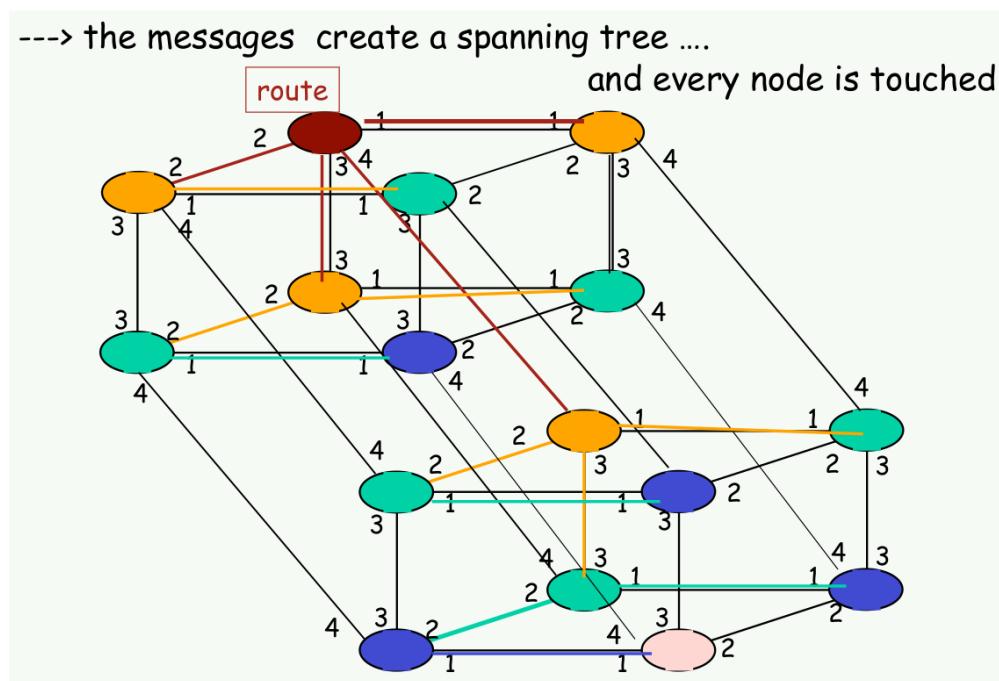
1. L'initiator invia il messaggio a tutti i suoi vicini
2. Un nodo che riceve il messaggio dal link  $l$ , lo invia solo sui links con etichetta  $l' < l$

**Protocollo Gerarchico** : L'initiator informa un nodo in ogni sub-hypercube di dimensione  $k' < k$ . A questo punto, il messaggio va in ognuno di essi.

**Correttezza** : Si basa sul fatto che **ogni nodo viene toccato (Spanning Property)**

Per ogni coppia di nodi  $x, y$  esiste un percorso di **etichette decrescenti**. Il messaggio verrà inviato su quel percorso.

In altre parole, gli archi usati formano un Sottografo di tip **Spanning** del Hypergrafo originale



La **message complexity** di questo algoritmo è quindi  $n - 1$ , e di conseguenza OTTIMALE

- Perchè ogni entità riceve le informazioni una sola volta

**dimostrazione :** Se così non fosse, allora gli **archi usati** formerebbero un ciclo contraddizione con le trasmissioni su percorsi con solo etichette decrescenti.

### ⌚ Tempo ideale >

Nel worst-case il tempo ideale è  $O(\log n)$  ed è OTTIMALE

## WakeUP Task

Cambiamo problema.

Abbiamo  $V$  insieme dei nodi

- **Configurazione iniziale :**
  - Sottoinsieme  $W \subset V$  di nodi che stanno nello stato **awake** e  $V - W$  nodi nello stato **asleep**.
  - Ogni nodo di  $W$  può iniziare una qualunque computazione (non abbiamo la restrizione  $UI$ )
- **Configurazione Finale** : Tutti i nodi in  $V$  devono essere nello stato **awake**

Le restrizioni standard sono  $R = \{TR, BL, CN\}$

## Protocollo WFLOOD

ASLEEP

Spontaneamente

    Invia (wake-up) a  $N(x)$

    Diventa AWAKE

Ricezione(wake-up)

    Invia (wake-up) a  $N(x) - \{\text{sender}\}$

    Diventa AWAKE

Il protocollo in questione ha :

- **msg\_comp** :  $\underbrace{2m - n + 1}_{(*)} \leq M(WFLOOD) \leq \underbrace{2m}_{(**)}$  (ottimale)
- **time ideal** :  $Time(WFLOOD) = D$  (ottimale)

$$(*) = \sum_{w \in W} |N(w)| + \sum_{v \in V-W} |N(v)| - 1 = 2m - n + 1$$

$$(**) = \sum_{v \in V} |N(v)| = 2m$$

Per quanto riguarda gli alberi :

$$M(WFLOOD/Tree) = n + k^* - 2, \text{ dove } k^* = \text{num. initiator}$$

Più  $k^*$  aumenta, più la complessità tende a  $n - 2$

## WakeUP Task su Grafi completi con Etichette uniche

In questa versione, abbiamo la restrizione che ogni nodo ha un nome univoco

### Ξ Teorema >

$$M(WFLOOD/K, Id) = O(n \log(n)), \text{ dove } K = \text{Clique}$$

Possiamo fare di meglio? la risposta è NO

$$\text{Vogliamo infatti dimostrare che } M(WFLOOD/K, Id) = \Omega(n \log(n))$$

### dimostrazione

Provare un lower-bound nel calcolo distribuito è un gioco a 2 : Protocollo PROT vs Avversario ADV

ADV, guarda il protocollo PROT, e decide:

1. le etichette dei nodi
2. quali sono gli initiator, e quando iniziano la computazione
3. **ritardo dei messaggi**
4. **combinare i link con l'etichettatura delle porte (per ogni sender)**

**Idea-Chiave** : ADV massimizza il num. di messaggi *inutili* tra tutti i nodi in stato AWAKE

- Gli strumenti che userà sono (3) e (4)

Diamo ora la definizione di sottoinsiemi connessi :

### Ξ Sottoinsiemi Connessi >

2 sottoinsiemi  $A, B$  si dicono **connessi** al tempo  $t$  se :

$$\exists a \in A \wedge b \in B : a \text{ e } b \text{ si scambiano messaggi al tempo } t$$

ADV sveglia la sorgente  $s$ . Quando  $s$  invia  $m(s)$ , ADV imposta l'etichetta  $l$  sull'arco  $e$  di  $y$ .

Usando (3)  $y$  non riceve  $m(s)$  finché  $y$  non invia  $m(y)$  su  $l'$ .

-> ADV assegna  $l'$  all'arco  $e$ .

Quindi, ADV consente a  $s$  e  $y$  di ricevere i messaggi **nello stesso momento**, quando sono già svegli.

Regole generali di ADV

Sia  $A = \{\text{nodi svegli}\}$  a un certo tempo  $t$

- **Goal di ADV** : i messaggi sono inviati ai nodi svegli e  $A$  è sempre connesso

Quando un nodo in stato AWAKE seleziona una porta UTILIZZATA, allora ADV non fa nulla, in quanto il messaggio è inutile.

Mentre...

Sia  $x \in A$  un nodo che esegue  $\text{send}(msg(x))$  su una **porta con etichettamento non assegnata**  $a$ . Allora ADV distingue due casisitche diverse :

1.  $x$  ha un'arco libero  $e$ , che collega un nodo AWAKE
  1. Allora ADV assegna  $a \rightarrow e$
2. Nessun collegamento libero. ADV crea un nuovo sottoinsieme di nodi  $B$  e poi lo collega ad  $A$  (nuovo STAGE)

Sia  $k$  il num. di nodi AWAKE

Partendo da  $A = \{x_0 = s, x_1, \dots, x_k\}$  si crea un sottoinsieme  $B$  di cloni ->  
 $B = \{z_0, \dots, z_k\} \implies |A| = |B|$

Sia  $x = x_i$  per un qualche  $i \implies m(x_i)$  verrà inviato a  $z_i$

ADV farà sì che la storia di  $B$  sia indipendente e identica a quella di  $A$  ( $x_i \sim z_i$ ) :

Per ogni  $i = 1, \dots, k : z_i$  si comporterà esattamente come  $x_i$ .

ma, quando  $z_i$  avrà utilizzato tutti i suoi collegamenti a  $B$  (**Fase 1**), allora il msg  $m(x_i)$  inviato a  $z_i$  **verrà ritardato** fino a che  $z_i$  non vorrà inviare  $m(z_i)$  su un'etichetta libera e tutti i collegamenti a  $B$  saranno stati utilizzati.

A quel punto (**Fase 2**), ADV assegna l'etichetta libera al collegamento  $z_i \iff x_i$

## Analisi dei costi

☒ Active(i) >

Sia  $Active(i) = \{\text{tutti i nodi in stato AWAKE allo stage } i\}$

$$New(i) = Active(i) - Active(i - 1)$$

**Fatto 1** :  $Active(0) = \{s\}$  e  $|New(i)| = |Active(i - 1)|$

⇒ **M(i-1)** >

$M(i - 1)$  = numero di messaggi scambiati prima dello stage  $i$

**Fatto 2** : ADV obbliga PROT (cioè i nodi attivi) a lasciare che i **nuovi** nodi scambino almeno lo stesso numero di messaggi dello stage  $(i - 1)$  prima di connettere  $A$  a  $B$ , cioè

$$\text{num. messaggi} = M(i - 1)$$

**Conseguenza** : **Numero totale di messaggi** prima della connessione  $A - B$  allo stage  $i$  è

$$\underbrace{M(i - 1)}_{\text{Stage } i} + \underbrace{M(i - 1)}_{\text{Stage } i \text{ prima di A-B}} = 2M(i - 1)$$

**Domanda principale** : Quanti messaggi PRIMA della fase  $i + 1$ ? (è necessaria l'analisi della fase 2)

Il numero esatto dipende dal PROT ma possiamo dare un Lower-bound

**Osservazioni.** ADV non inizia un nuovo STAGE finché

- C'è  $x \in Active(i)$  che invia  $m(x)$  su una nuova etichetta  $l$
- Tutti i collegamenti da  $x$  ad  $Active(i)$  non sono liberi

**Conseguenza.**  $x$  ha comunicato con tutti i nodi in

$$Active(i) = Active(i - 1) \cup New(i)$$

Due casi:

- $x \in Active(i - 1)$ . Allora, i msg  $x \iff New(i)$  avviene nella Fase  $i$ 
  - Quindi, dopo la connessione (Fase 2)

$$\text{num. msg(i)} \geq |New(i)| = |Active(i - 1)|$$

- $x \in New(i)$ . Allora tutti i messaggi stanno in fase  $i$ 
  - Ancora

$$\text{num. msg(i)} \geq |New(i)| = |Active(i - 1)|$$

- Combinando otteniamo

$$M(i) \geq \underbrace{2M(i - 1)}_{\text{Fase 1}} + \underbrace{|Active(i - 1)|}_{\text{Fase 2}}$$

Abbiamo quindi che

$$|Active(i)| = \begin{cases} 1 & i = 0 \\ 2|Active(i-1)| & i \geq 1 \end{cases}$$

Di conseguenza, per un  $i$  generico vale che  $|Active(i)| = 2^i$

E quindi :

$$M(i) \geq 2M(i-1) + 2^{i-1} \geq i \cdot 2^{i-1} \underset{\substack{\sqsubseteq \\ \text{Num. Stage} = \log(n)}}{=} \Omega(n \log(n)) \quad \blacksquare$$

Abbiamo quindi dimostrato che il lower-bound della message complexity per il protocollo WFLOOD su *clique* ( $K$ ) con etichettamento dei nodi (ID) è pari a  $\Omega(n \log(n))$

# Lezione 5 ADRC - Spanning Tree Construction, Protocollo SHOUT & SHOUT+

Fin'ora abbiamo visto il problema Broadcast, andando a vedere diversi protocolli che lo risolvono.

Spostiamoci ora su un'altro problema, ovvero il problema **Spanning Tree Construction**

## Spanning Tree Construction

### ↳ Spanning Tree >

Uno spanning tree  $T$  di un grafo  $G = (V, E)$  è un sottografo aciclico di  $G$  tale che  $T = (V, E')$  e  $E' \subset E$

Le assunzioni che andiamo a fare in questo problema sono le seguenti :

- **Single Initiator**
- Bidirectional Links
- Total Reliability
- $G$  connesso.

## Motivazioni

Perchè vogliamo costruire uno spanning tree del sistema distribuito?

Ci sono due motivazioni :

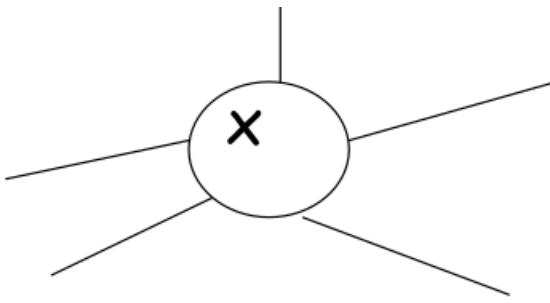
1. Effettuare comunicazioni (**broadcast**) su una sottorete è molto più efficiente ( $m$  è più piccolo)
2. La subnet deve essere :
  1. Spanning (cioè toccare tutti i nodi della rete)
  2. Connected

Ogni **Spanning Tree** è lo spanner connesso avente **minimo** numero di links

## Il problema ST Distribuito

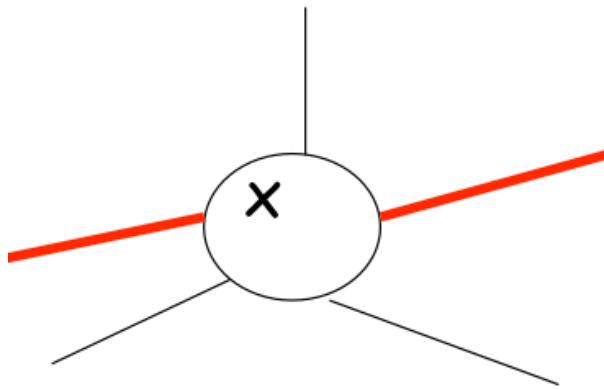
**Configurazione iniziale** :  $\forall x, Tree - \text{neigh}(x) = \{\}$

- Significa che all'inizio, ogni nodo ha variabile Tree-neigh vuota
- La situazione è la seguente ↓



**Configurazione finale :**  $\forall x, \text{Tree} - \text{neigh}(x) = \{\text{x-links che appartengono allo ST}\}$

- Quindi, alla fine, ogni nodo dovrà selezionare solo gli archi che fanno parte della soluzione globale, ovvero l'intero ST del sistema distribuito
- La situazione è la seguente ↓



#### ⚠️ Osservazione cruciale >

I nodi possono **non conoscere** lo ST anche **dopo** la computazione

Vediamo ora i vari protocolli per questo problema

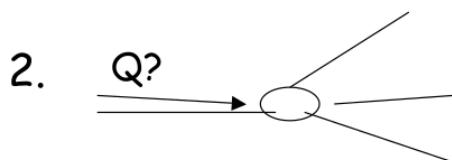
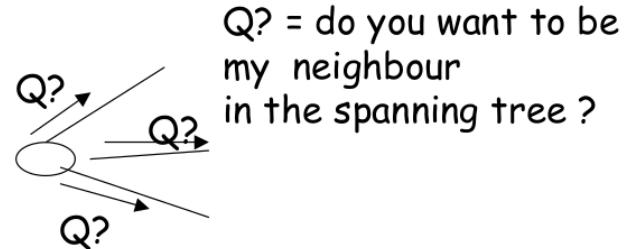
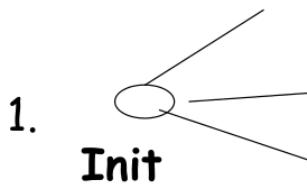
## Protocollo SHOUT

L'idea è : Come fa un nodo  $x$  a selezionare i suoi vicini validi per ST?

Semplicemente **lo chiede** !

La risposta dei vicini sono : Rispondi sempre SI alla prima richiesta e NO alle altre

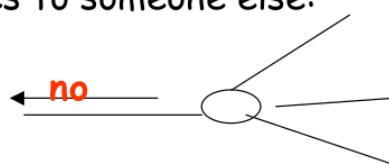
L'initiator inizia a chiedere e poi il protocollo procede come il **broadcast** con gli **acks**



If it is the **first time**:



If I have already answerd yes to someone else:



Vediamo ora il protocollo effettivo

Gli stati possibili

- $S = \{\text{Initiator}, \text{Idle}, \text{Active}, \text{Done}\}$
- $S_{\text{init}} = \{\text{Initiator}, \text{Idle}\}$
- $S_{\text{term}} = \{\text{Done}\}$

#### INITIATOR

Spontaneamente

```
root=True
Tree-neigh = {}
invia (Q) a N(x)
counter = 0
diventa ACTIVE
```

#### IDLE

```
Riceve(Q)
root=False
parent = sender
Tree-neigh = {sender}
invia (Yes) a sender
```

```

counter = 1
if counter = |N(x)|
    diventa DONE
else
    invia (Q) a N(x)-{sender}
    diventa ACTIVE

```

ACTIVE

Riceve(Q)  
 invia (No) a sender

Riceve(Yes)  
 Tree-neigh = Tree-neigh U sender  
 counter = counter+1  
 if counter = |N(x)|  
 diventa DONE

Riceve(No)  
 counter = counter+1  
 if counter = |N(x)|  
 diventa DONE

**oss :** SHOUT=FLOOD+REPLY

## Terminazione di SHOUT

### ⇒ Lemma >

Sotto le assunzioni standard  $R$  su  $G$ , si ha che  $\forall s \in V, \forall v \in V - \{s\}$ , dopo l'esecuzione del protocollo SHOUT  $\exists p : s \rightarrow v$  tale che :

1. Da  $s \rightarrow v$  è passato il messaggio  $Q$ ?
2. Da  $v \rightarrow s$  è passato il messaggio  $Yes$

Questo cammino  $\exists$  per le assunzioni TR e CN, in più c'è sempre un ordine.

$\nexists$  ciclo perchè un nodo dovrebbe aver risposto  $Yes$  a **due** "padri", e questo non è possibile

- Possono arrivare anche 1000 messaggi  $Q$ ?, ma le risposte saranno gestite una per una (ad es. usando una coda)

## Correttezza di SHOUT

La correttezza del protocollo SHOUT si basa su queste assunzioni :

- Se  $x \in \text{Tree} - \text{neigh}(y) \implies y \in \text{Tree} - \text{neigh}(x)$
- Se  $x$  invia Yes a  $y$ , allora  $x \in \text{Tree} - \text{neigh}(y)$  ed è connesso all'initiator tramite una **catena** di Yes-links
- Ogni  $x$  (eccetto l'initiator) invia esattamente un singolo Yes -> (no cicli !!)

Allora, lo spanning graph definito dalle relazioni Tree-neigh risulta essere **connesso, aciclico** e contiene **tutti** i nodi.

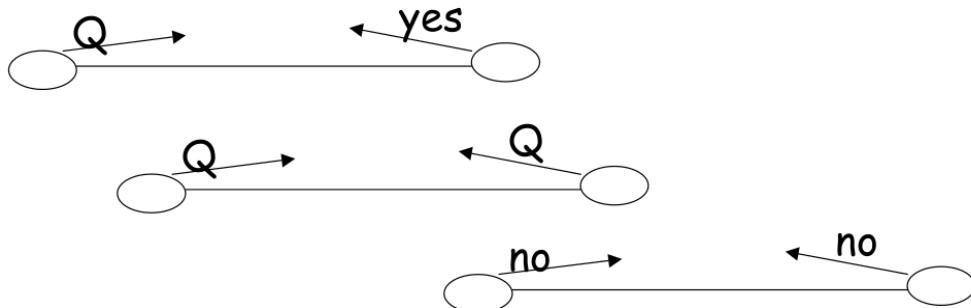
## Message Complexity

In modo informale,

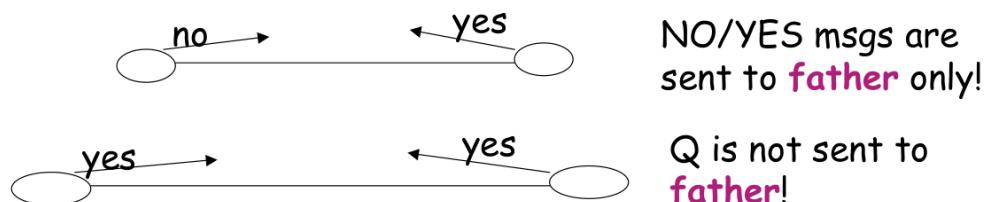
$$M(\text{SHOUT}) = 2M(\text{FLOOD})$$

Nel dettaglio :

**Situazioni possibili :**

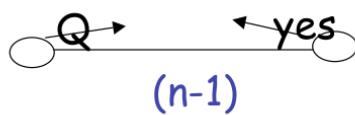


**Situazioni impossibili :**

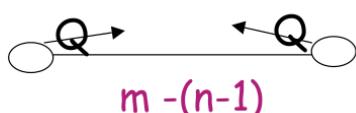


Nel worst-case abbiamo che :

**Numero totale di messaggi Q?**



only one Q on the **ST** links



on the other links

Totalle :

$$2(m - (n - 1)) + (n - 1) \implies 2m - n + 1$$

**Numero totale di NO :**



- $2(m - (n - 1))$

**Numero totale di YES :**



- Esattamente  $(n - 1)$

In totale :

$$2m - n + 1 + 2(m - (n - 1)) + n - 1 = 4m - 2n + 2$$

Quindi  $M(SHOUT) = 4m - 2n + 2$

Notiamo che  $\Omega(M)$  è **lower-bound** anche in questo caso

Cerchiamo di abbassare la complessità  $4m$ , raffinando il protocollo SHOUT

## Protocollo SHOUT+

I messaggi NO sono realmente necessari? la risposta è no

- un nodo ha bisogno di sapere quando può **terminare**

**oss :**

Se  $y$  manda No a  $x$ , allora deve accadere che  $y$  ha mandato anche  $Q?$  a  $x$  PRIMA.  
Quindi, ricevere  $Q?$  può essere *interpretato* come un NO dalla stessa porta !!

Vediamo quindi il protocollo

Gli stati possibili

- $S = \{Initiator, Idle, Active, Done\}$
- $S_{init} = \{Initiator, Idle\}$
- $S_{term} = \{Done\}$

INITIATOR

Spontaneamente

root=True

Tree-neigh = {}

invia (Q) a N(x)

```
counter = 0
diventa ACTIVE
```

IDLE

```
Riceve(Q)
    root=False
    parent = sender
    Tree-neigh = {sender}
    invia (Yes) a sender
    counter = 1
    if counter = |N(x)|
        diventa DONE
    else
        invia (Q) a N(x)-{sender}
        diventa ACTIVE
```

ACTIVE

```
Riceve(Q) (da interpretare come N0)
    counter = counter+1
    if counter = |N(x)|
        diventa DONE

Riceve(Yes)
    Tree-neigh = Tree-neigh U sender
    counter = counter+1
    if counter = |N(x)|
        diventa DONE
```

## Message Complexity

Su ogni link ci saranno esattamente 2 messaggi :

Questo :



Oppure questo :



Di conseguenza,

$$M(SHOUT+) = 2m$$

Che risulta essere **migliore** di  $4m - 2n + 2$

**Terminazione locale** : Ogni nodo sa quando può smettere di lavorare

**Terminazione globale** : Può un qualunque nodo decidere **quando** tutti gli altri devono terminare??

- Possibile solo sotto l'assunzione UI, pagando un prezzo per la message complexity pari a  $n - 1$  in più

## Costruzione ST usando Broadcast

### ☒ Teorema >

L'esecuzione di ogni protocollo di Broadcast costruisce uno ST

### idea della dimostrazione

Definiamo il *father* di un nodo  $x$

### ☒ Father(x) >

Il Father di un nodo  $x$  è definito come :

$$Father(x) = [\text{il nodo che per primo invia il messaggio Init a } x]$$

**Fatto** : l'ordine parziale  $Father(x) > x$  definisce uno ST con radice nell'Initiator

**oss** : Solo un Broadcast non è sufficiente (Non abbiamo la conoscenza locale dello ST)

## SPT : Considerazioni Finali

Diamo 2 considerazioni finali :

1. Lo ST calcolat (costruito) **dipende** dal Protocollo
2. Lo ST calcolat (costruito) **dipende** dall'esecuzione del Protocollo

Prendiamo per esempio il protocollo SHOUT+

Ogni possibile ST del grafo  $G$  può essere l'output di SHOUT -> "basta gestire i delay dei msg"

Di conseguenza :  $diam(ST) \gg diam(G)$  (Problema)

E quindi sorge un nuovo problema, ovvero costruire uno ST  $T^*$  tale che  $diam(T^*) \simeq diam(G)$

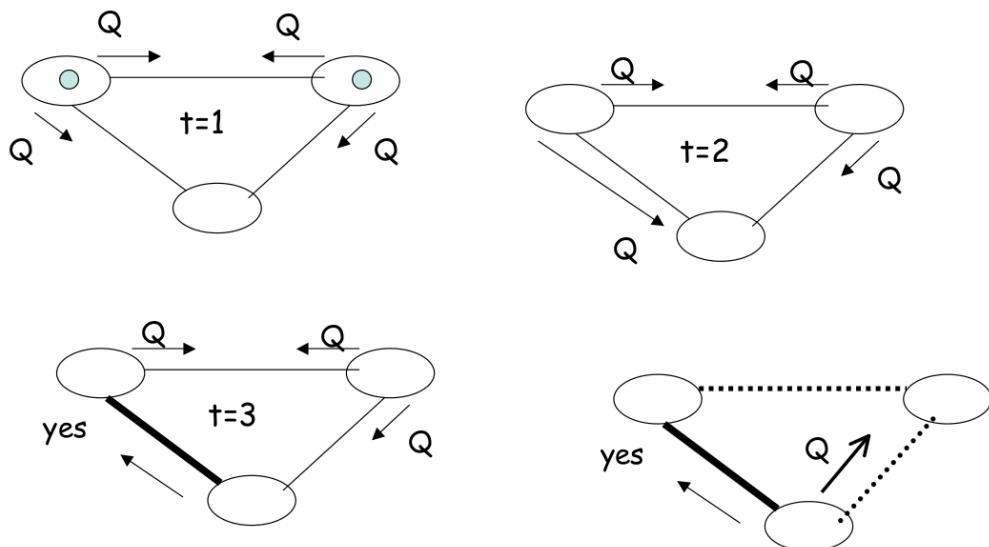
## Costruzione ST usando Multiple Initiator

Le domande che ci poniamo in questa situazione sono :

1. Possiamo costruire lo ST usando più Initiator?
2. E se usassimo il protocollo SHOUT+ per questo?
3. Possiamo creare un protocollo **deterministico** per questo problema?

La risposta a tutte e 3 le domande è NO.

Vediamo un esempio più teorema dopo



### ☒ Teorema >

Il problema del costruire lo ST è **deterministicamente** irrisolvibile sotto le assunzioni  $R$

**dim**

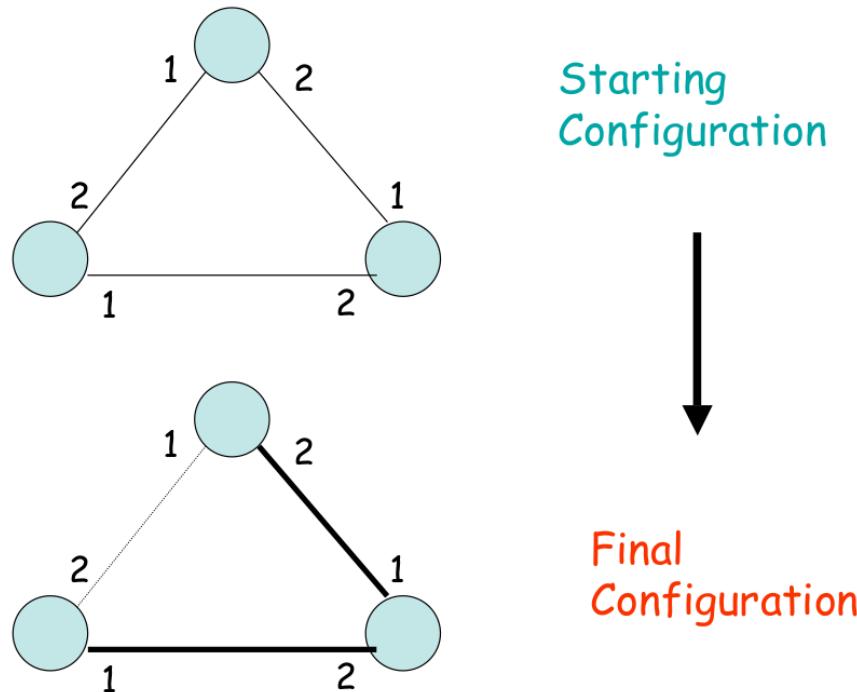
Mettiamoci nel caso più semplice, ovvero 3 nodi  $x, y, z$  che formano un triangolo, che hanno gli stessi **stati iniziali**.

- tutti iniziano la computazione a tempo  $t = 0$
- Eseguizioni sincrone

**Fatto** : Ad ogni istante di tempo  $t > 0$ , i 3 nodi devono :

- trovarsi nello **stesso stato**
- fare le **stesse cose**

Quindi, sarà **impossibile** per loro selezionare 2 links su 3



# Lezione 6 ADRC - Depth First Traversal, Protocollo BACK & DFT+, Tecnica di Saturazione & Minimum Finding

Spostiamoci ora nel problema noto come **Depth First Traversal (DFT)**

## Problema Depth First Traversal

In questo problema andiamo a vedere la capacità di **somministrare** un token (risorsa) ad ogni nodo del SD, in modo mutualmente esclusivo (ad ogni istante di tempo, uno e un solo nodo avrà il token)

Ci mettiamo sotto le assunzioni standard  $R = UI, BL, TR, CN$

Vediamo la prima versione del protocollo per DFT, chiamato protocollo Back

## Protocollo Back

Per ogni nodo  $x \in V$

1. Quando visitato per la **prima volta** - ricorda chi è il **(padre)**
  1. Invia il token a uno dei vicini non visitati
  2. aspetta la risposta
2. Quando il vicino riceve il **token**
  1. Se già visitato, rimanda il token al padre usando un **back-links**
  2. Altrimenti, **Io invia in modo sequenziale a tutti** i vicini non visitati prima di rimandarlo indietro
3. Se non ci sono più nodi inesplorati, ritorna il token (**reply**) al padre
4. Alla ricezione di un reply, invia il token ad un'altro vicino non visitato

## Message Complexity

Abbiamo 3 tipi di messaggi : token, back, return

- I messaggi di tipo Token vengono sempre inviati almeno una volta ->  $O(m)$
- O vengono inviati o i messaggi Return (se in stato idle quando ricevo il token) **oppure** i messaggi back ->  $O(m)$

Su ogni arco quindi passano due copie di messaggi, perchè return e back sono esclusive (o una o l'altra)

Quindi

$$M(Back) = 2m = O(m)$$

Notiamo che  $\Omega(m)$  è un lower-bound anche qui

## Time Complexity (Ideal Time)

Ricordiamo che per fare l'analisi della time complexity dobbiamo metterci nel mondo sincrono, impostando i ritardi tutti a "1".

L'algoritmo per DFT, ovvero Back, è **sequenziale**. Quindi :

$$Time(Back) = M(DFS) = \Theta(m)$$

 **Lower Bound generale** >

$$Time(DFT) = \Omega(m)$$

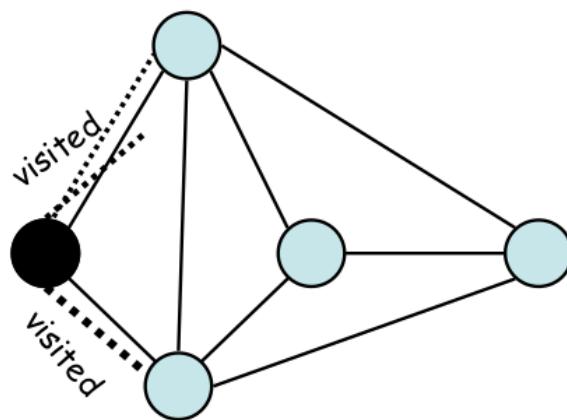
Possiamo migliorare questo algoritmo? SI, migliorando il tempo ideale

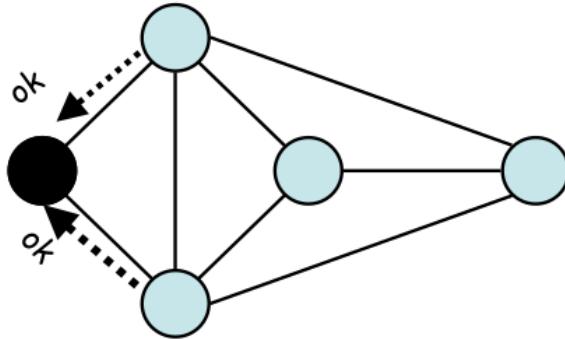
## Protocollo DFT+

**oss** : In grafi densi, il più dei messaggi potrebbe essere sui back-links, e di conseguenza il più del tempo è speso su questi archi

Dovremmo riuscire ad evitare l'invio di messaggi sui back-links

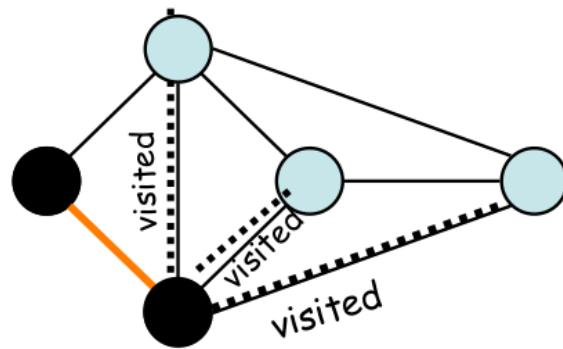
**Idea** : Il nodo  $x$  informa in parallelo || tutto  $N(x)$  quando viene visitato



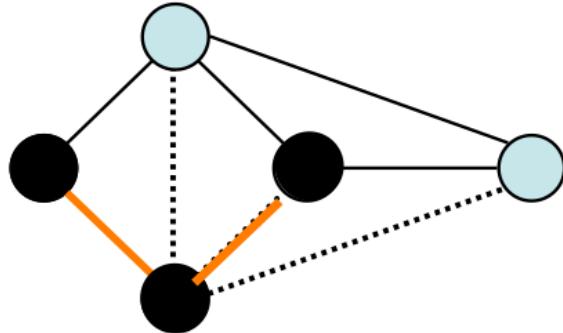
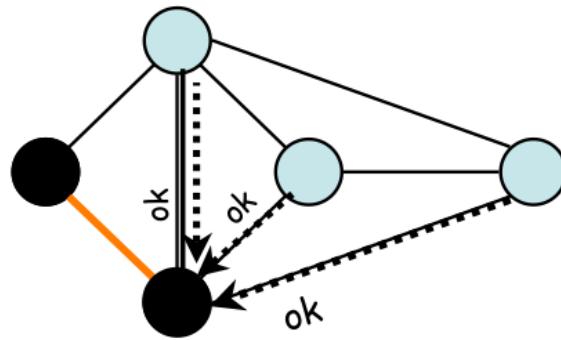


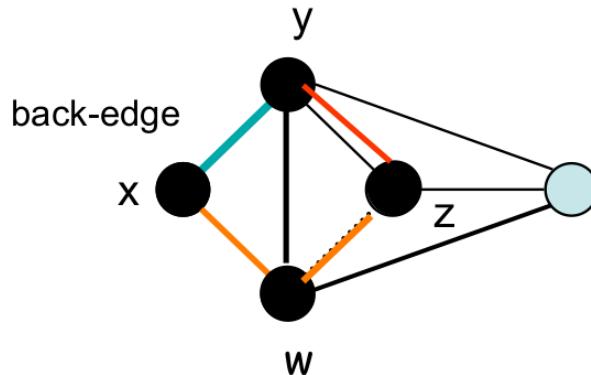
$N(x)$  invia il messaggio **ok** indietro, sempre in ||

- Questo è necessario quando il sistema non è sincrono



Dopo aver ricevuto **tutti gli akcs**, il nodo  $x$  invia il token a uno dei suoi vicini **non visitati** ( $x$  già li conosce tutti)





Il nodo  $y$  sa che  $x, w$  sono stati già visitati e che  $z$  è suo padre, quindi i back-links vengono scoperti in **parallelo**

## Message Complexity

I messaggi sono : Token, Return, Visited, Ack (OK)

Ogni entità (tranne l'initiator) : riceve 1 token, invia 1 return  $\rightarrow 2(n - 1)$

Ogni entità invia 1 Visited a tutti i suoi vicini tranne che al sender

- Sia  $s$  l'initiator, allora

$$|N(s)| + \sum_{x \neq s} (|N(x)| - 1) = 2m - (n - 1)$$

- Stessa cosa per gli Ack :  $2m - (n - 1)$

Quindi in totale

$$M(DFT+) = 4m - 2(n - 1) + 2(n - 1) = 4m$$

Notare che abbiamo peggiorato la message complexity da Back a DFT+

## Time Complexity

Se il sistema è sincrono :

- Token e Return sono inviati **sequenzialmente** :  $2(n - 1)$
- Visited e Ack sono inviati in **parallelo** :  $2n$

Totale

$$Time(DFT+) = 4n - 2$$

Il tempo è **lineare** in  $n$  e non in  $m$  (TOP)

## Computazioni in Alberi

Di seguito elencate alcune tra le tecniche più importanti quando si parla di computazioni su alberi.

- Saturazione
- Minimum Finding
- Eccentricità
- Center
- Ranking (non faremo)

Vediamo ora la Saturation Technique

## Saturation Technique

Le assunzioni sono :  $R = \{BL, OM, TR, KT\}$ , dove  $OM$  = Ordered Message e  $KT$  = Knowledge of the Topology

L'ultima assunzione ci afferma che i nodi sanno di essere **foglie**, oppure **nodi interni**

Vediamo ora la tecnica in questione.

Gli stati disponibili sono  $S = \{Available, Awake, Processing\}$

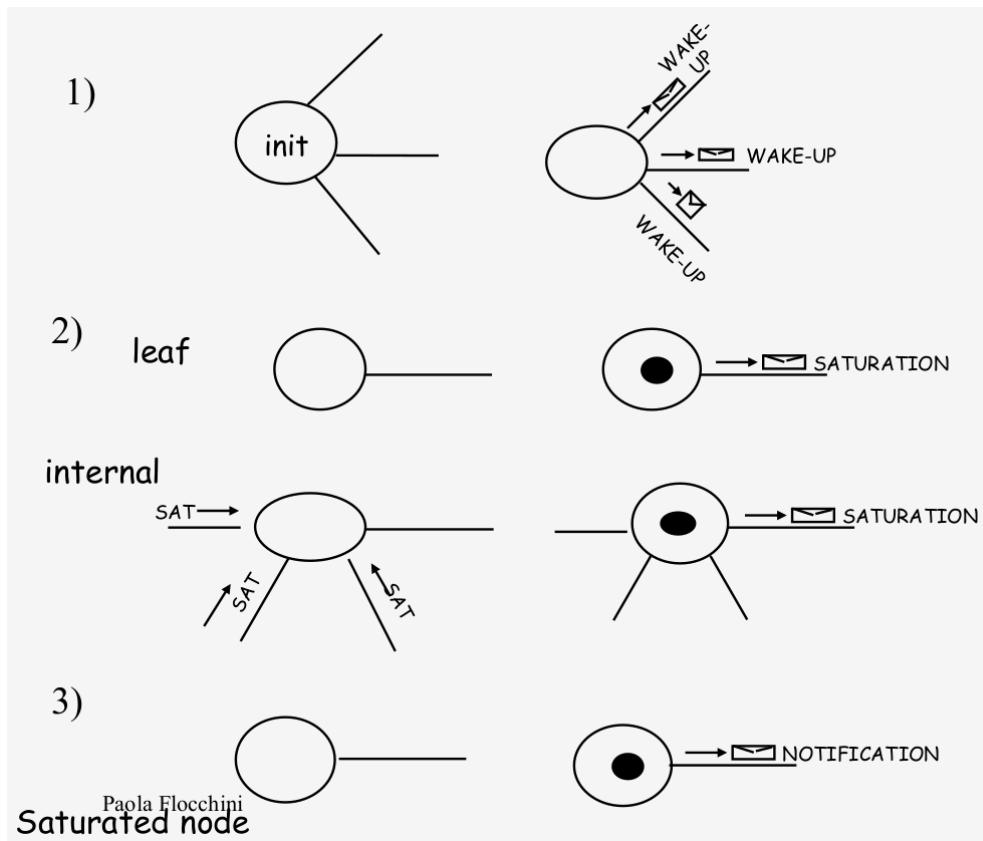
- All'inizio, tutte le entità stanno in stato **Available**
- Entità arbitrarie possono iniziare la computazione (**Multiple Initiator**)

**Goal :**

- Tutti i nodi devono stare in stato **Awake**
- Due nodi adiacenti devono essere selezionati (**link election**)
- I due nodi **selezionati** sono pronti per iniziare una qualunque computazione

Le fasi della tecnica sono :

- **Activation Phase** : Iniziata dagli **initiators** : Tutti i nodi sono attivati (questo non è altro che il WakeUP)
- **Saturation Phase** : Inizia dalle **foglie** : Una coppia **univoca** di vicini viene identificata (Anche detti nodi saturati)
- **Resolution Phase** : Una generica computazione iniziata dai **nodi saturati**



Vediamo ora il protocollo

Gli stati sono

- $S = \{Available, Active, Processing, Saturated\}$
- $S_{init} = Available$

AVAILABLE (Non sono stato ancora attivato)

Spontaneamente

```

Invia (Activate) a N(x)
Vicini = N(x)
if |Vicini| = 1 : /*questo if è caso speciale se il nodo è FOGLIA*/
    M = ("Saturation")
    parent = Vicini
    invia (M) a parent
    diventa PROCESSING(saturation)
else:
    diventa ACTIVE

```

AVAILABLE (Non sono stato ancora attivato)

Ricevo(Activate)

```

invio (Activate) a N(x)-{sender}
Vicini = N(x)
if |Vicini| = 1 : /*sono FOGLIA*/
    M = ("Saturation")

```

```

parent = Vicini
invia (M) a parent
diventa PROCESSING /*Saturation Phase*/
else:
    diventa ACTIVE

```

ACTIVE (Non ho ancora iniziato la fase di saturazione)

Ricevo(M)

```

Vicini = Vicini - {sender}
if |Vicini| = 1 :
    M = ("Saturation")
    parent = Vicini
    invia (M) a parent
    diventa PROCESSING

```

PROCESSING (Ho già iniziato la fase di saturazione e sto ricevento M dal mio parent)

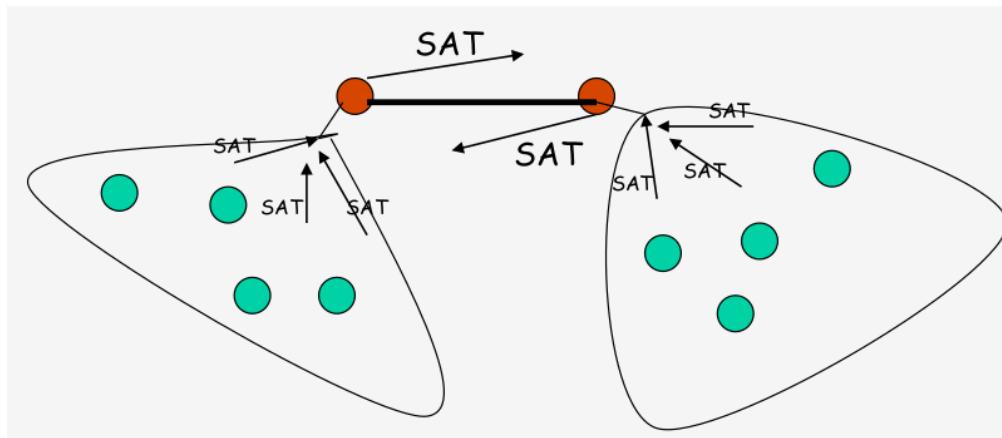
Ricevo(M)

```
divento SATURATED
```

Vediamo ora un teorema molto importante

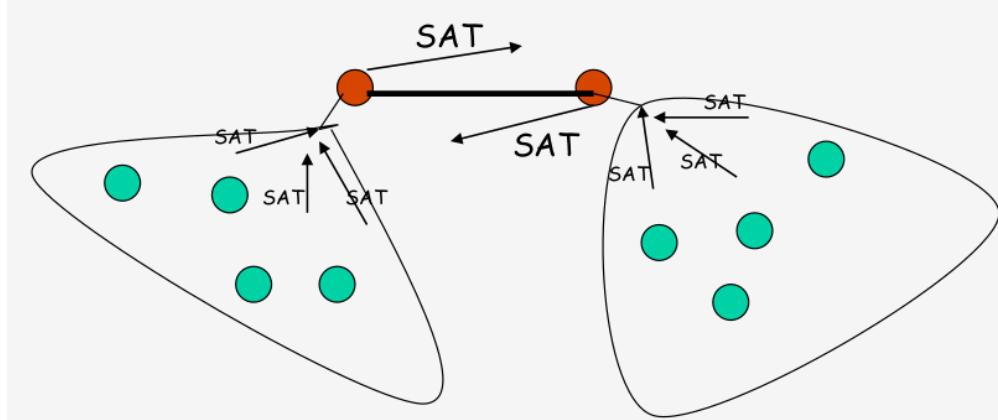
### ☞ Teorema >

Esattamente due nodi in stato PROCESSING diventeranno SATURATED, e loro sono vicini



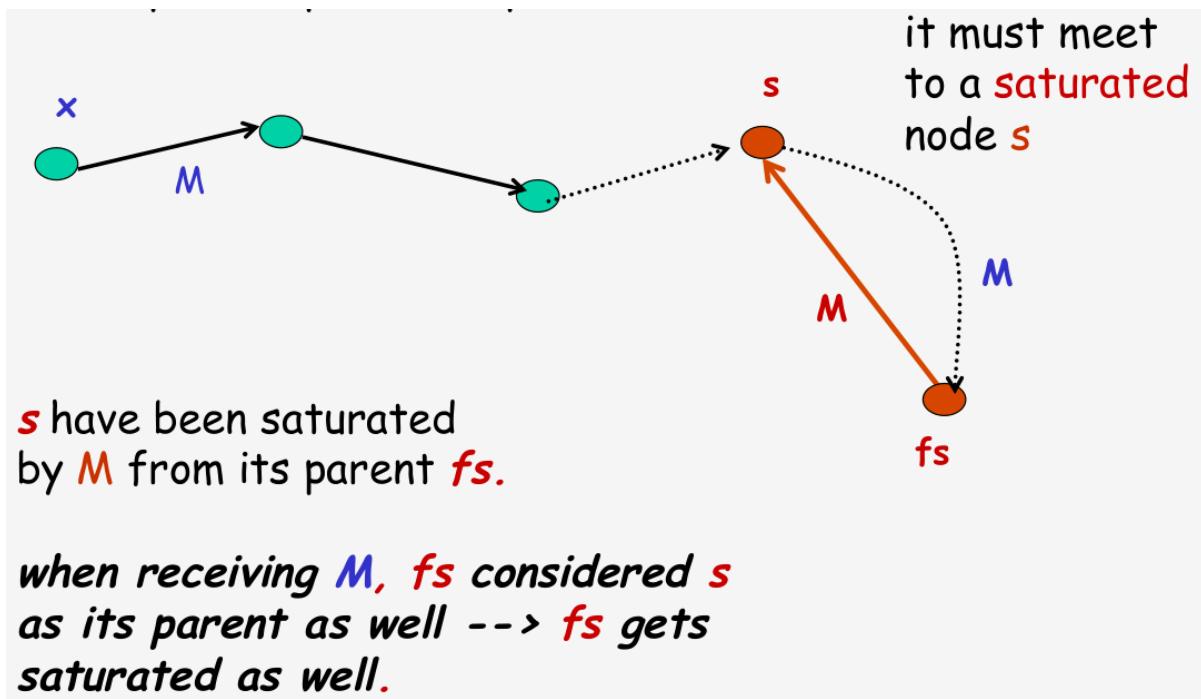
**Dim :**

Un nodo PROCESSING solo **dopo** aver inviato il messaggio "saturato" al proprio parent



Un nodo diventa SATURATED solo dopo aver ricevuto un messaggio nello stato PROCESSING dal suo parent

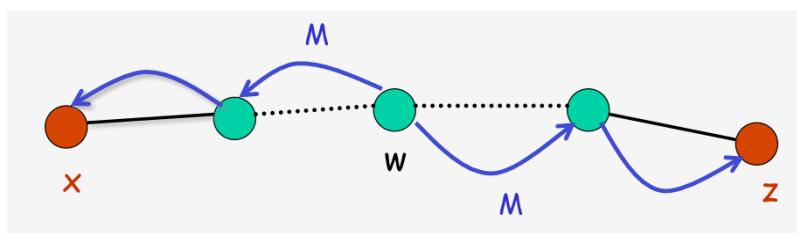
Ora, scegliamo un nodo  $x$  e consideriamo il percorso  $M$ -path che va indietro verso il suo parent, parent del parent, etc...



Perchè esattamente due nodi (adiacenti)?

Assumiamo che ce ne siano 3 :  $x, y, z$

Allora, in un ALBERO, 2 di loro devono essere **non collegati**. Quindi, consideriamo il percorso fra questi due

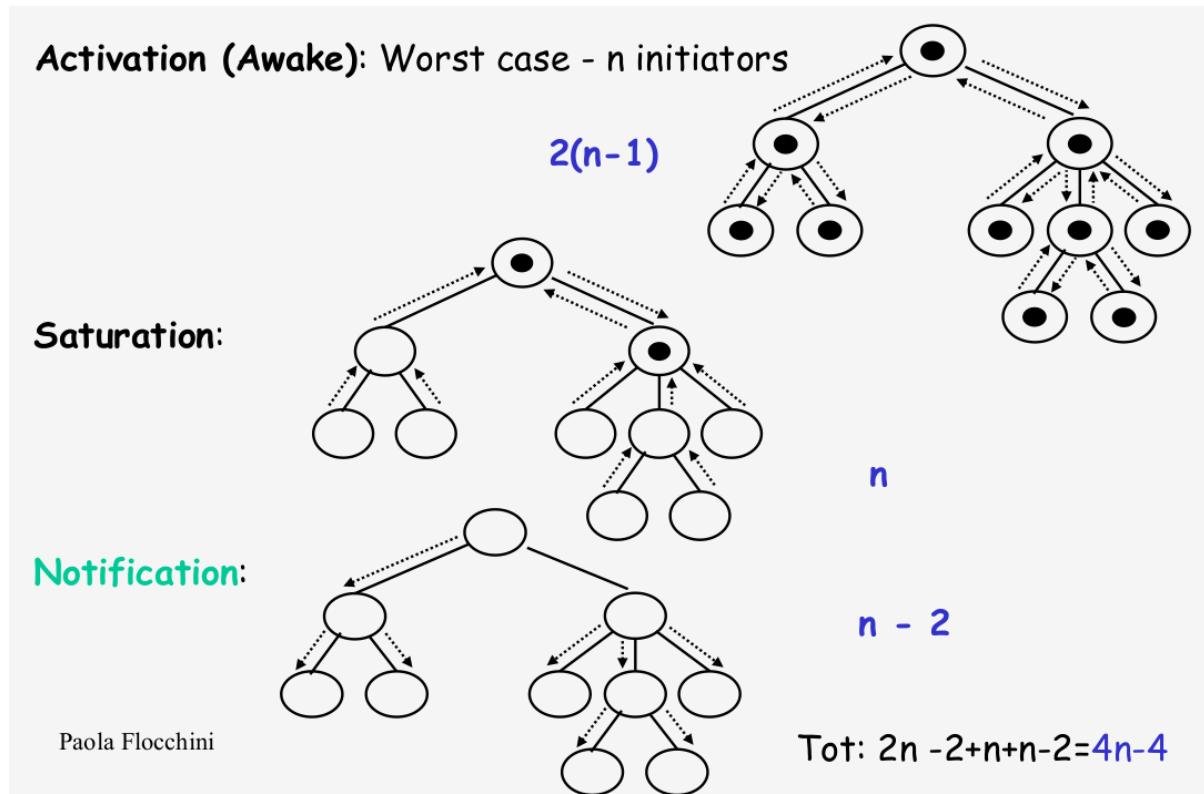


Non è possibile che  $w$  invii il messaggio  $M$  indietro sia al nodo  $x$  che al nodo  $y$  ■

Per concludere, quali nodi diventeranno **saturati** dipende solo dai ritardi imprevedibili

## Message Complexity (Alberi)

Caso con  $n$  initiator



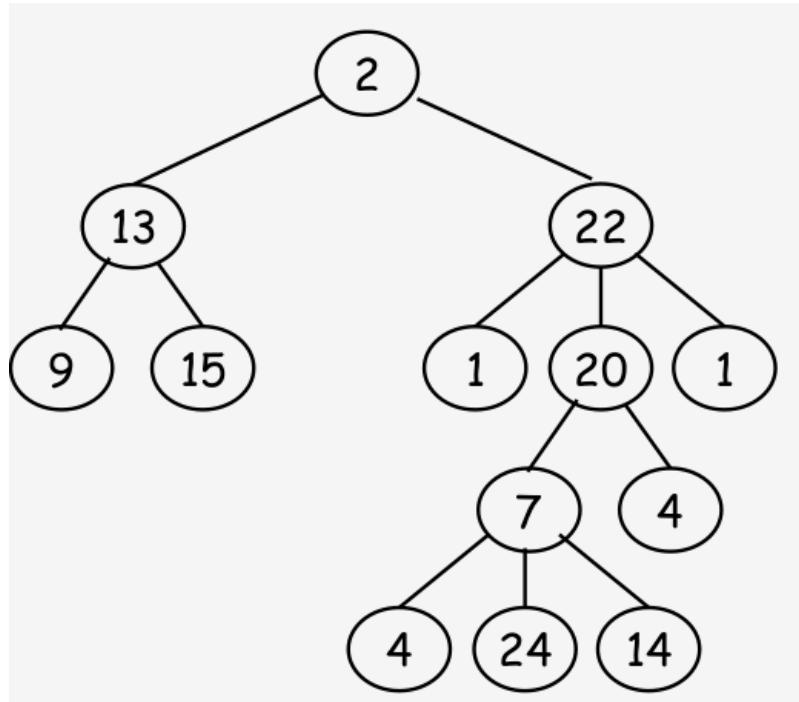
In generale, con  $k^*$  initiator

- Activation :  $n + k^* - 2$
- Saturation :  $n$
- Notification :  $n - 2$
- Totale

$$2n + k^* - 2$$

## Mettere informazioni nel messaggio di saturazione - Problema Minimum Finding

### Minimum Finding



**Inizio :** Ogni entità  $x$  ha in input un valore  $val(x)$

**Fine :** Alla fine, ogni entità deve sapere se è il minimo oppure no

**oss :**

In un albero radicato, il problema Min-Find è di facile risoluzione

Usiamo le assunzioni :

- figlio->parent (**Convergecast**)
- parent->figlio (**Broadcast**)

Gli step sono :

1. Broadcast (Da root=initiator=leader alle foglie)
2. Convergecast (da foglie a root)
3. Broadcast (notifica del minimo globale)

In un albero normale invece, usiamo la tecnica della Saturazione

1. Il ruolo della root è sostituito dai due nodi saturati (**co-leader**)
2. Le foglie iniziano mandando il messaggio di saturazione  $M$  contenente il loro valore  $val(x)$
3. Quando un nodo interno riceve **tutti** i messaggi  $M$  dai figli, calcola il valore del minimo e lo invia sopra al proprio parent
4. I due co-leader a questo punto sapranno chi è il minimo. Quindi lo inviano a tutti tramite Broadcast

Vediamo ora il protocollo

Gli stati sono :

- $S = \{Available, Active, Processing, Saturated\}$

- $S_{init} = Available$

AVAILABLE

Spontaneamente(Initiators)

```

    invia (Activate) a N(x)
    min = val(x)
    Vicini = N(x)
    If |Vicini| = 1 :
        M=("Saturation",min) /*Sono foglia*/
        parent = Vicini
        invia (M) a parent
        diventa PROCESSING
    Else:
        diventa ACTIVE

```

AVAILABLE

Ricevo(Activate)

```

    invio (Activate) a N(x)-{sender}
    min = val(x)
    Vicini = N(x)
    If |Vicini| = 1 :
        M=("Saturation",min) /*Sono foglia*/
        parent = Vicini
        invia (M) a parent
        diventa PROCESSING
    Else:
        diventa ACTIVE

```

/\*fase bottom-up : Saturazione verso i co-leaders\*/

ACTIVE

Ricevo(M)

```

    min = MIN{min,M}
    Vicini = Vicini-{sender}
    If |Vicini| = 1 :
        M = ("Saturation",min)
        parent = Vicini
        invia (M) a parent
        diventa PROCESSING

```

PROCESSING

Ricevo(M) /\*Sono co-leader\*/

```

    min=MIN{min,M}

```

```

Notifica = ("Resolution",min)
invia (Notifica) a N(x)-parent
If val(x)=min :
    diventa MINIMUM
Else :
    diventa LARGE

```

## PROCESSING

```

Ricevo(Notifica) /*Broadcast e Update*/
    invia (Notifica) a N(x)-parent
    If val(x)=Valore_Ricevuto :
        diventa MINIMUM
    Else :
        diventa LARGE

```

Abbiamo visto come la Saturazione può essere applicata al calcolo del minimo, ma non serve solo a questo.

Infatti la Saturazione può essere usata per calcolare una classe generale di funzioni in alberi distribuiti.

Quale però? La risposta è la seguente :

La classe di funzioni  $F$  deve essere un **semi-gruppo**, che rispetta le seguenti proprietà :

1. Associativa ->  $F(a, b, c) = F(F(a, b), c)$
2. Commutativa ->  $F(a, b, c) = F(b, c, a)$

Infatti, per queste operazioni la Saturazione funziona correttamente :

*Minimo, Massimo, Somma, Prodotto, Predicati Logici*

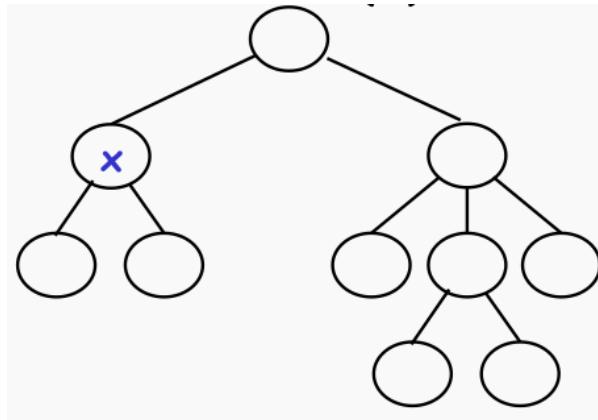
# Lezione 7 ADRC - Finding Eccentricities & Center Finding

## Finding Eccentricities

Il problema che vogliamo studiare qui è quello di trovare l'**eccentricità** di ogni entità  $x$  nel SD con topologia ad albero

Vediamo che, l'eccentricità di un nodo  $x$  è definita come :

$$r(x) = \max_{y \in V} \{d(x, y)\}, \quad d(x, y) = \text{distanza dal nodo } x \text{ a } y$$



Notiamo che nell'esempio  $r(x) = 4$

Vediamo alcune topologie particolari :

1. Clicque :  $r(x) = 1, \forall x$

2. Stella :

$$r(x) = \begin{cases} 1 & x \text{ è al centro} \\ 2 & \text{Altrimenti} \end{cases}$$

3. Catena :

$$r(x) = \begin{cases} \frac{n}{2} & x \text{ è al centro} \\ \leq \frac{n}{2} & \text{se } x \text{ non sta al centro e non sta agli estremi} \\ n & x \text{ è uno dei due estremi} \end{cases}$$

Ora, come facciamo a trovare l'eccentricità di ogni nodo?

## Idea 1 (Banale)

1. Ogni nodo effettua una richiesta (broadcast)
2. Le foglie inviano un messaggio per collezionare le distanze (Convergecast)

La messagge complexity di questo protocollo è  $O(n^2)$

Abbiamo troppi messaggi **ridondanti**

Possiamo riciclare qualche valore??

## Idea 2 (Migliore usando Saturazione)

Anche qui usiamo la Saturation Technique

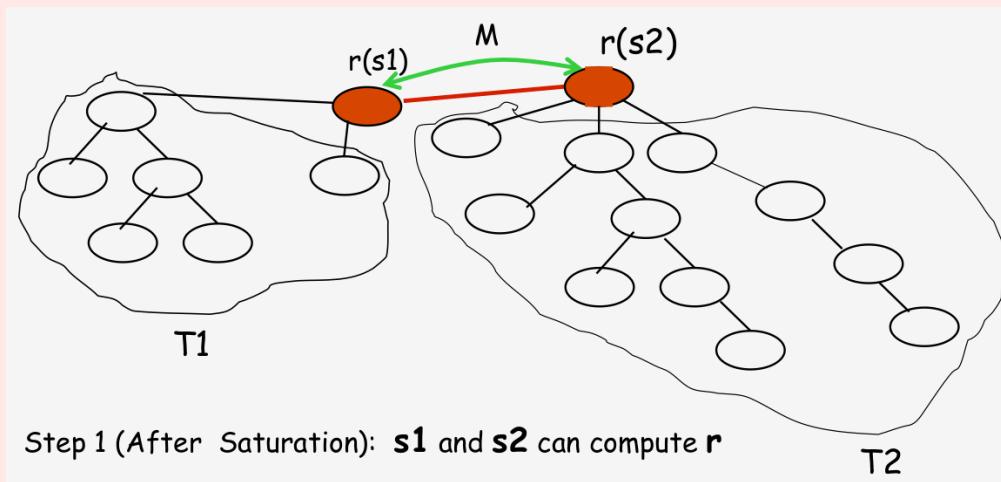
1. Trova l'eccentricità dei due nodi saturati usando il messaggio  $M$  (fase di saturazione)
2. Propaga le informazioni **necessarie** in modo che gli altri nodi possano calcolarsi la loro eccentricità (nella fase di notifica)

La messagge complexity di questo protocollo vediamo essere la stessa della Saturazione

Vediamo ora qualche teorema

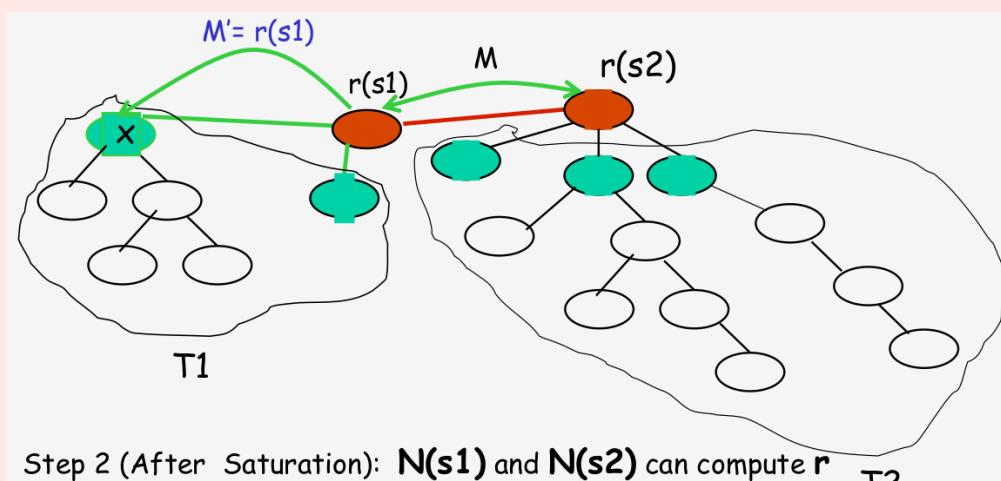
### ☒ Step 1 >

$$r(s_1) = \max\{h(T_1), 1 + r(s_2)\}$$



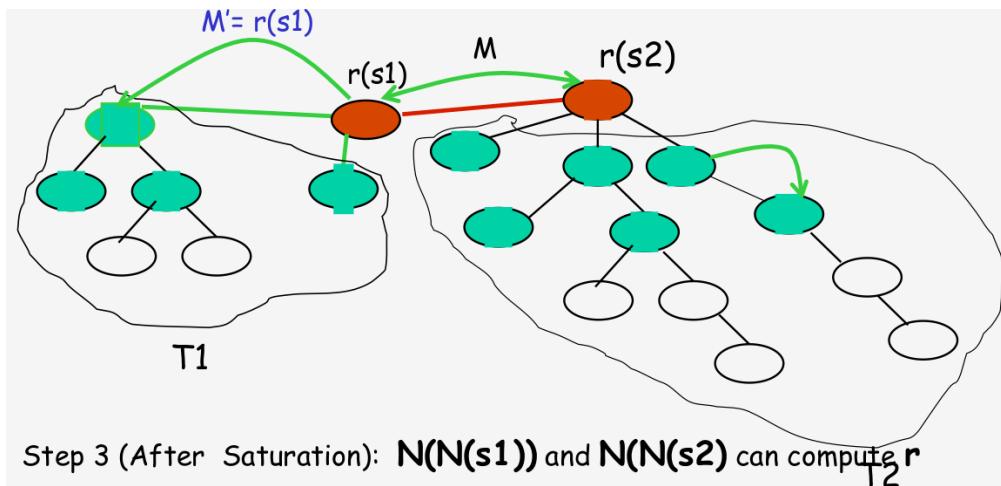
### ☒ Step 2 >

$$r(x) = F(knw(x), M' + 1)$$



Ricordiamo che  $knw(x)$  è il valore di eccentricità del nodo  $x$  relativa al suo sottoalbero (e la conosce grazie alla prima passata di saturazione)

E così via



Vediamo ora il protocollo

## Protocollo

Gli stati sono :

- $S = \{Available, Active, Processing, Saturated\}$
- $S_{init} = Available$

L'idea del protocollo è :

- Definisci una variabile locale **maxdist(y)** che prende la **Depth** del sottoalbero radicato in  $y$
- Quando calcolata,  $y$  invia  $maxdist(y)$  al suo parent
- Quando un nodo  $x$ , in stato PROCESSING, riceve dal suo parent il messaggio  $M = (\|Saturation\|, maxdist)$  (quindi  $x$  diventa saturato) allora :
  - $x$  potrà calcolare la sua **eccentricità**
  - $x$  potrà inviare le **informazioni corrette** ad ognuno dei suoi figli :
    - $x$  invia a  $y \rightarrow \max\{maxdist(z) | z \# y \text{ è figlio di } x\}$

Il protocollo è :

**definisci Distance[]**

AVAILABLE

Spontaneamente

Invia (Activate) a  $N(x)$

```

Distance[x]=0
Vicini = N(x)
If |Vicini| = 1 :
    maxdist = 1 + Max{Distance[*]}
    M=("Saturation",maxdist)
    parent = Vicini
    Invia (M) a parent
    diventa PROCESSING
Else :
    diventa ACTIVE

Invia (Activate) a N(x) - {sender}
Distance[x]=0
Vicini = N(x)
If |Vicini| = 1 :
    maxdist = 1 + Max{Distance[*]}
    M=("Saturation",maxdist)
    parent = Vicini
    Invia (M) a parent
    diventa PROCESSING
Else :
    diventa ACTIVE

```

ACTIVE

```

Ricevo(M)
    Distance[{sender}] = Distanza_Ricevuta
    Vicini = Vicini - {sender}
    If |Vicini| = 1 :
        maxdist = 1 + Max{Distance[*]}
        M=("Saturation",maxdist)
        parent = Vicini
        Invia (M) a parent
        diventa PROCESSING

```

PROCESSING

```

Ricevo(M)
    Distance[{sender}] = Distanza_Ricevuta
    r(x) = Max{Distance[z]:z appartiene a N(x)}
    For all y appartenente a N(x)-{parent} do :
        maxdist = 1 + Max{Distance[z] : z appartiene a N(x)-{y}}
        invia ("Resolution",maxdist) a y

```

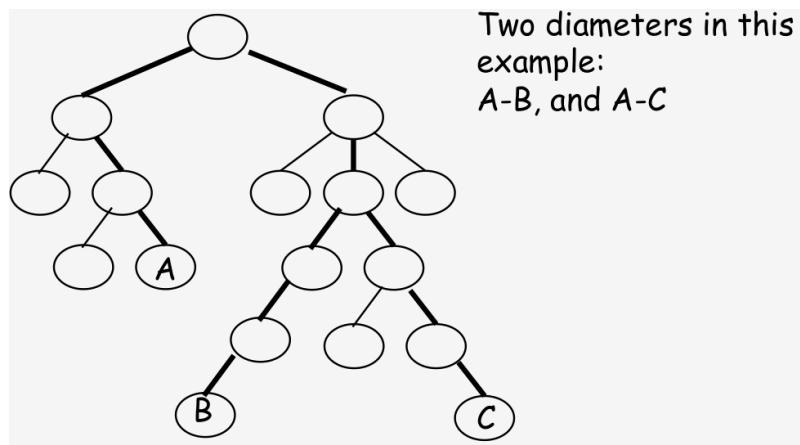
EndFor

diventa DONE

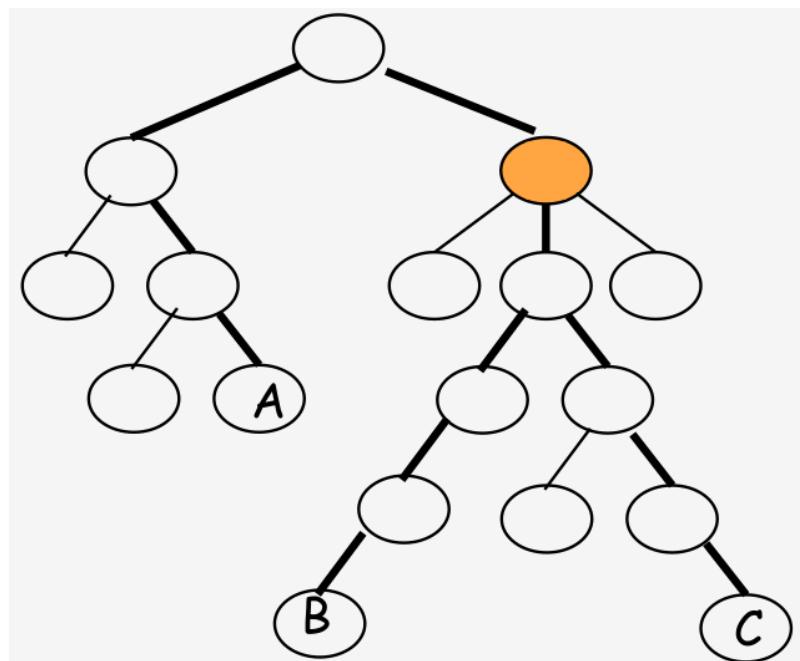
## Center Finding

Qui quello che ci chiediamo è trovare il **centro** dell'albero, ovvero quel nodo  $x$  tale che  $r(x) \leq r(y), \forall y \in V - \{x\}$ . Per fare ciò, dobbiamo prendere, tra tutte le distanze massime, quella minimizzata

Introduciamo quindi il concetto di **percorso diametrale**, che in qualche modo può essere associato al longest path



Il centro è il nodo con eccentricità minima



Un'idea potrebbe essere :

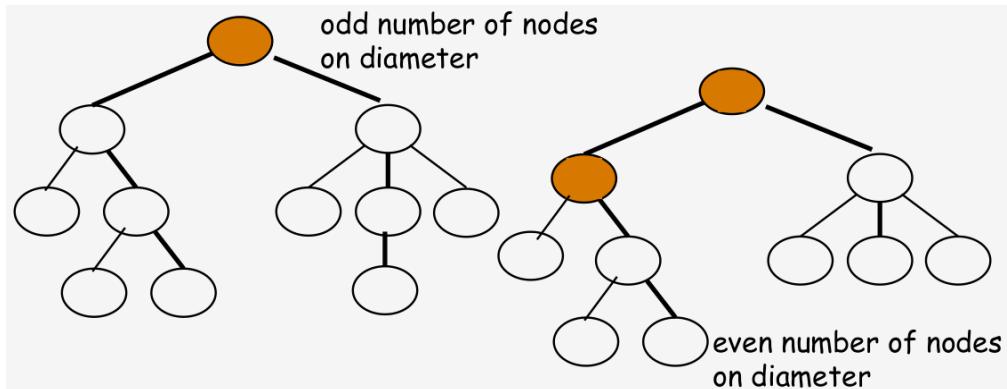
1. Trovare tutte le eccentricità  $\rightarrow 4n - 4$
2. Trovare la più piccola  $\rightarrow 2n - 2$

### 3. Totale $6n - 6$

Una strategia migliore può essere trovata studiando le proprietà che il nodo centro ha.

#### ☒ Proprietà 1 >

In un albero c'è un unico centro, oppure due centri che sono però vicini



Come vediamo, se ci sono un numero **pari** di nodi sul diametro allora abbiamo 2 centri, altrimenti un centro solo.

#### ☒ Proprietà 2 >

I centri si trovano sui percorsi diametrali

Prima di definire la terza proprietà, definiamo :

$$d_1[x] = \text{max distance}, \quad d_2[x] = \text{seconda max distance} \quad (\text{attraverso vicini differenti})$$

#### ☒ Proprietà 3 >

Un nodo  $x$  è il centro  $\iff d_1[x] - d_2[x] \leq 1$   
 (se  $d_1[x] = d_2[x]$  allora c'è un solo centro)

A questo punto, l'idea per il Center Finding diventa :

1. Trova tutte le eccentricità
2. Ogni nodo può capire localmente se è il centro o no
3. Totale :  $4n - 4$

**Un'altra idea migliore :**

1. Trova l'eccentricità dei nodi saturati  $\rightarrow 3n - 2$
2. Localmente controllo se sono il centro

3. Se non lo sono, propago l'informazione relativa alla distanza **solo** in direzione del centro

$$\rightarrow \leq \frac{n}{2}$$

4. Totale :  $\leq 3.5n - 2$

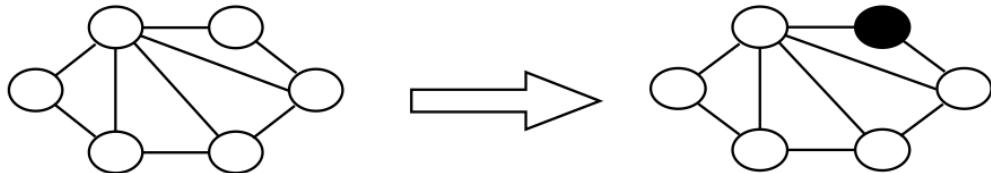
# Lezione 8 ADRC - Leader Election, Protocollo All The Way, Protocollo AsFar, Stage Technique, Architettura Mesh

## Leader Election

Nei SD spesso è necessario che una singola entità coordini il lavoro delle altre entità per la risoluzione dei task.

Tale entità prende il nome di **leader**, e trovare il leader in un insieme di entità è noto come il **Leader Election Problem**.

Fare Leader Election significa "Rompere" la simmetria



La configurazione iniziale è :  $\forall x \in V, state(x) = asleep$

- quante sono le possibili configurazioni iniziali?  $\rightarrow 2^{n-1}$   
La configurazione finale è :  
 $\exists!x \in V : state(x) = leader \wedge \forall y \in V - \{x\} : state(y) = follower$
- quante sono le possibili configurazioni finali?  $\rightarrow n$

Le restrizioni che usiamo sono  $R = \{TR, BL, CN\}$

Notiamo come non abbiamo messo la restrizione *UI*, dato che ogni entità può essere il leader.

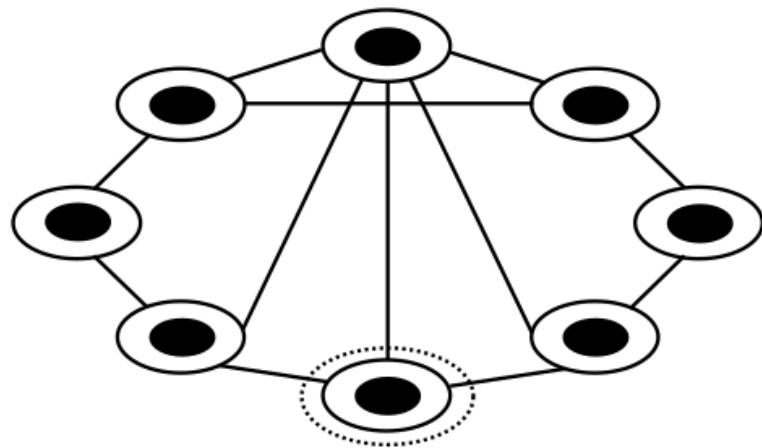
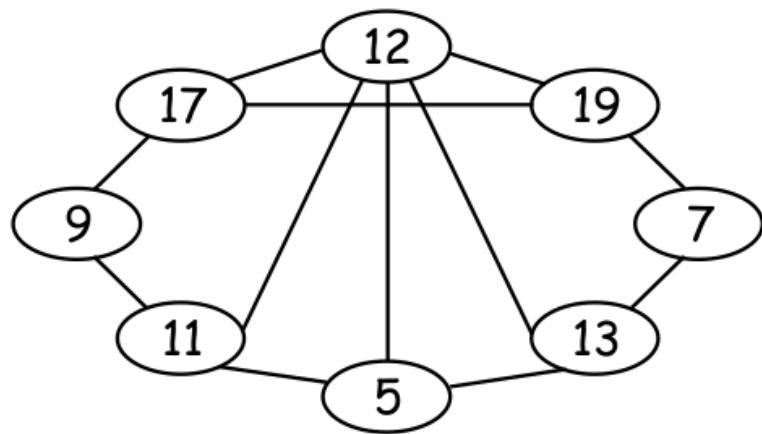
Allora vale il seguente teorema

### ✖ Angluin 80 >

Il problema dell'**elezione** non può essere risolto in modo deterministico se le entità nel SD non hanno **identità** differenti.

La dimostrazione è identica a quella del triangolo, nel problema STP

**oss** : con differenti ID, il problema Minimum Finding diventa una elezione



## Elezione in alberi

Ad ogni nodo  $x$  è associato un identificatore distinto  $v(x)$

Un semplice algoritmo per fare leader election è :

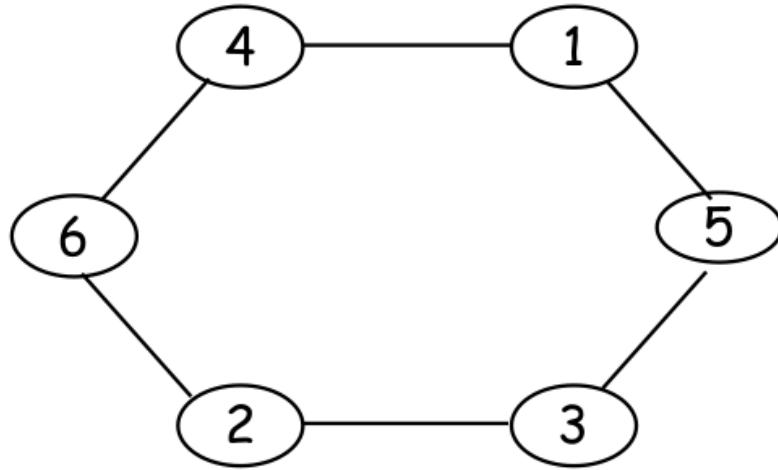
1. Esegui la tecnica della saturazione
2. Scegli il nodo saturato che ha il valore minimo

## Elezione in grafi ad anello

Cos'è un grafo ad anello? Un grafo ad anello è un grafo in cui :

- $n$  entità
- $m = n$  links
- Topologia simmetrica
- Ogni entità ha esattamente due vicini
- C'è il senso della direzione (sx,dx)

Ad esempio



Notiamo che gli ID dei nodi non sono necessariamente **consistenti**, cioè non è detto che  $id(x_i) = i$  o  $id(x_i) = k \implies id(x_{i+1}) = k + 1$

### definizione del problema

- configuzioni iniziali ->  $\forall x, state(x) = idle$ , un nodo (o più) viene svegliato
  - il num di config. iniziali è  $2^n$
- configuzione finale ->  $\exists!v \in V : state(v) = leader \wedge \forall x \in V - \{v\} : state(x) = follower$ 
  - il num di config. finali è  $n$

## Protocollo All The Way

**Idea base** : Ogni  $id$  viene trasmesso circolarmente nell'anello -> ogni entità vede tutte le identità

Assunzioni :

- Ci sono due versioni -> link unidirezionali/bidirezionali
- Orientazione locale -> non necessariamente presente il senso della direzione
- Identità distinte

Quando un nodo decide di fermarsi?

- Risp 1 : Quando "finalmente" riceve il suo messaggio indietro
- Risp 2 : Quando riceve esattamente  $n - 1$  altre identità

Entrambe le risposte sono però sbagliate

1. Assume un'ordinamento dei messaggi (FIFO)
2. Assume che i nodi conoscano  $n$

Vediamo ora il protocollo

Gli stati sono :

- $S = \{ASLEEP, AWAKE, FOLLOWER, LEADER\}$
- $S_{init} = ASLEEP$
- $S_{term} = \{FOLLOWER, LEADER\}$

## INITIALIZE

```
count = 0
size = 1
know = false
invia("Election",id(x),size) a destra
min = id(x)
```

## ASLEEP

Spontaneamente

```
    INITIALIZE
    diventa AWAKE
```

Riceve("Election",value,counter)

```
    INITIALIZE
    invia indietro ("Election",value,counter+1) agli altri
    min = Min{min,value}
    count=count+1
    diventa AWAKE
```

## AWAKE

```
Ricevo("Election",value,counter)
if value != id(x) :
    invia("Election",value,counter+1) agli altri
    min = Min{min,value}
    count=count+1
    if known = true : /*solo se known=true conosco n e posso fare
CHECK e TERMINATION*/
        CHECK
    endif
else :
    ringsize=counter
    known = true
    CHECK
endif
```

CHECK

```

if counter=ringsize :
    if min = id(x) : /*posso fermarmi qui*/
        diventa LEADER
    else
        diventa FOLLOWER
    endif
endif

```

## Message Complexity

Ogni entità attraversa ogni link  $\rightarrow n^2$

La grandezza di ogni messaggio è  $\log(id + counter)$

Quindi abbiamo :

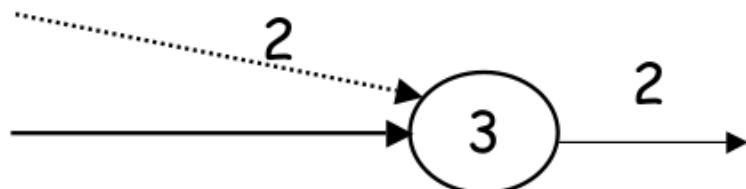
- $O(n^2)$  messaggi
- $O(n^2 \log(\text{MaxID}))$  bits necessari

**oss** : Il protocollo funziona sia per i link unidirezionali che per quelli bidirezionali

## Protocollo AsFar (as it can)

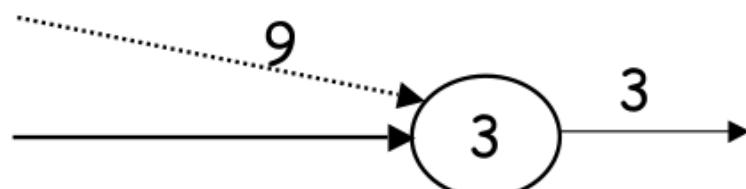
**Idea base** : Non è necessario mandare e ricevere messaggi con id **più grandi** degli id che sono stati già visti

Assunzioni le stesse dell'altro protocollo



Ricevo  $y$  **più piccolo** di me :

- **invio(y)** agli altri vicini



Ricevo *y più grande* di me :

- **invio(x)** agli altri vicini (se non è stato già inviato)

Vediamo il protocollo

Gli stati sono gli stessi dell'altro protocollo

ASLEEP

Spontaneamente

```
invia("Election",id(x)) a destra
min = id(x)
diventa AWAKE
```

Ricevo("Election",value)

```
invia("Election",id(x)) a destra
min = id(x)
if value < min : /*questo può essere evitato se id(x) > value */
    invia("Election",value) agli altri
    min = value
endif
diventa AWAKE
```

AWAKE

Ricevo("Election",value)

```
if value < min :
    invia("Election",value) agli altri
    min = value
else
    if value = min :
        NOTIFY
    endif
endif
```

Ricevo(Notify)

```
invio (Notify) agli altri
diventa FOLLOWER
```

NOTIFY

```
invia(Notify) a destra
diventa LEADER
```

## Correttezza e Terminazione

Il **Leader** sa che lui è Leader quando riceve ***il suo messaggio*** indietro.

Quindi lui può TERMINARE, ma quando gli altri sanno che possono terminare?

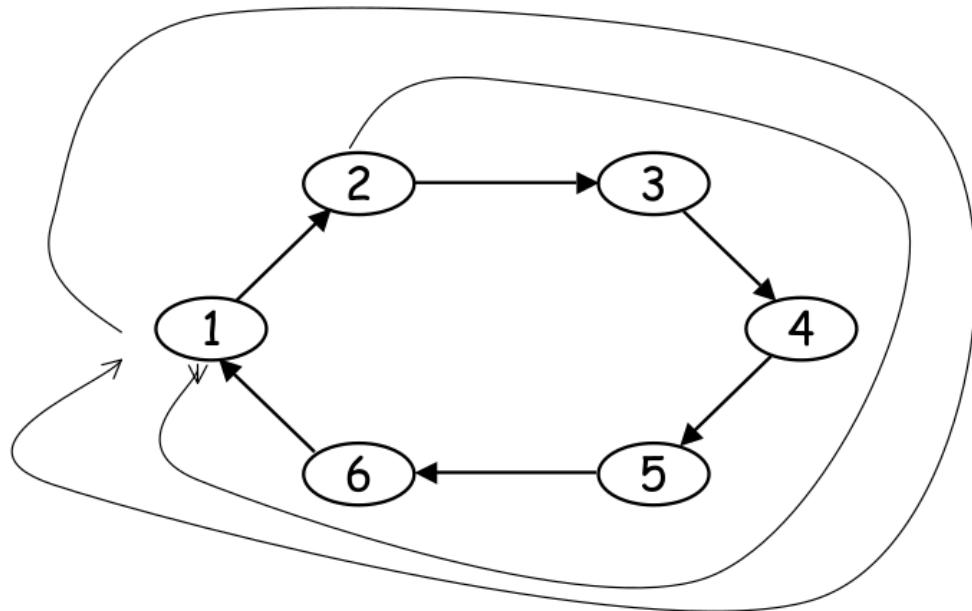
- è necessaria una **Notifica** dal Leader

## Message Complexity

Dipende fortemente dalla configurazione degli ID sull'anello

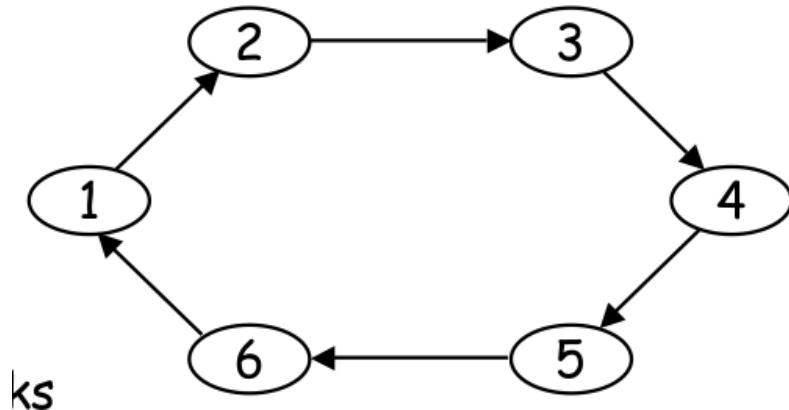
$$\text{rank}(id) = \text{num. id più piccoli di id} + 1$$

La situazione worst-case è :



## Analisi Worst-Case

Gli ID sono distribuiti in sequenza, da sx verso dx



Abbiamo che :

- Dal nodo 1  $\rightarrow n$  links
- Dal nodo 2  $\rightarrow n - 1$  links

- Dal nodo 3 →  $n - 2$  links
- :
- Dal nodo  $n \rightarrow 1$  link

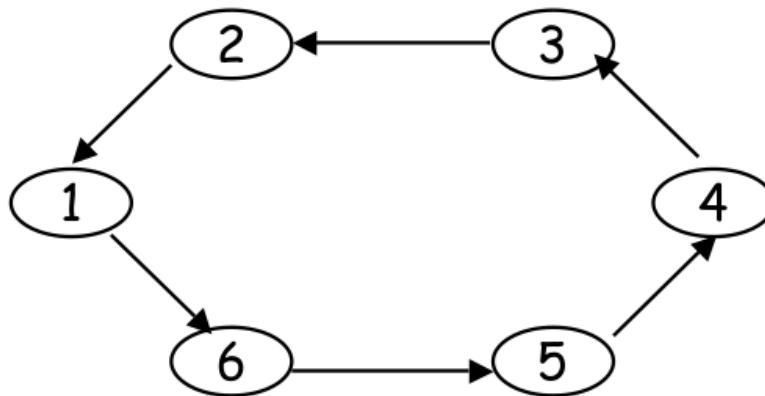
In totale quindi abbiamo :  $n + (n - 1) + (n - 2) + \dots + 1 = \sum_{i=1}^n \frac{n(n+1)}{2}$

Quindi abbiamo che :

$$M(AsFar)_{worst-case} = \frac{n(n+1)}{2} + \underbrace{n}_{\text{notifica}}$$

Leggermente migliore del protocollo All The Way

### Analisi Best-Case



- Dal nodo 1 →  $n$  links
- $\forall i \neq 1 \rightarrow 1$  link (totale  $n - 1$ )

In totale quindi :

$$M(AsFar)_{best-case} = n + (n - 1) + \underbrace{n}_{\text{notifica}} = O(n)$$

## Leader Election : Stage Technique

Il protocollo lavora in *Stages* -> Ogni nodo sa in quale Stage lui sta performingo

**Analisi del protocollo** : Alla fine di ogni Stage, alcune **proprietà globali** vengono mantenute

## Controlled Distance

**Idea base** : Si lavora in Stages. Un'entità mantiene il controllo sul suo messaggio

Assunzioni :  $\{BL, ID, LO\}$

Il senso della direzione è mantenuto solo per semplicità, non serve

Cosa ci serve?

1. Distanza limitata (per evitare che messaggi grandi viaggino troppo a lungo) -> stage  $i$  distanza  $2^{i-1}$
2. Messaggi di ritorno (se viene visto qualcosa più piccolo non si continua)
3. Controllo su ambo i lati
4. Il più piccolo vince sempre

Il protocollo lavora così -> I *candidati* iniziano l'algoritmo

Vediamo lo Stage  $i$ , per qualche  $i$

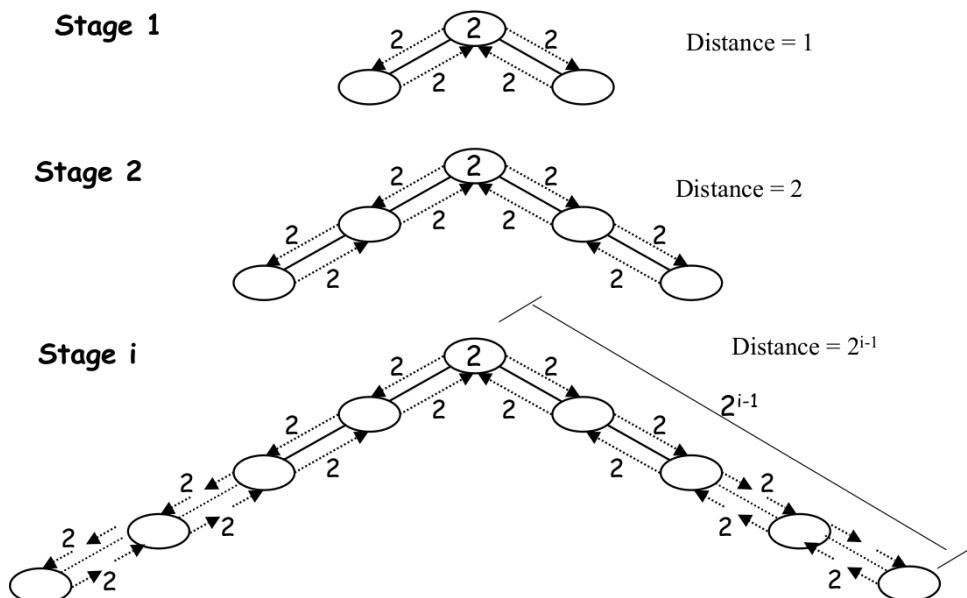
1. Ogni candidato invia un messaggio con il proprio ID in entrambe le direzioni
2. Il messaggio viaggerà finchè non incontrerà un ID più piccolo o finchè non raggiungerà una certa distanza ( $2^{i-1}$ )
3. Se il messaggio non incontra un ID più piccolo, allora tornerà indietro al proprietario.  
(chiamati **messaggi di feedback**)



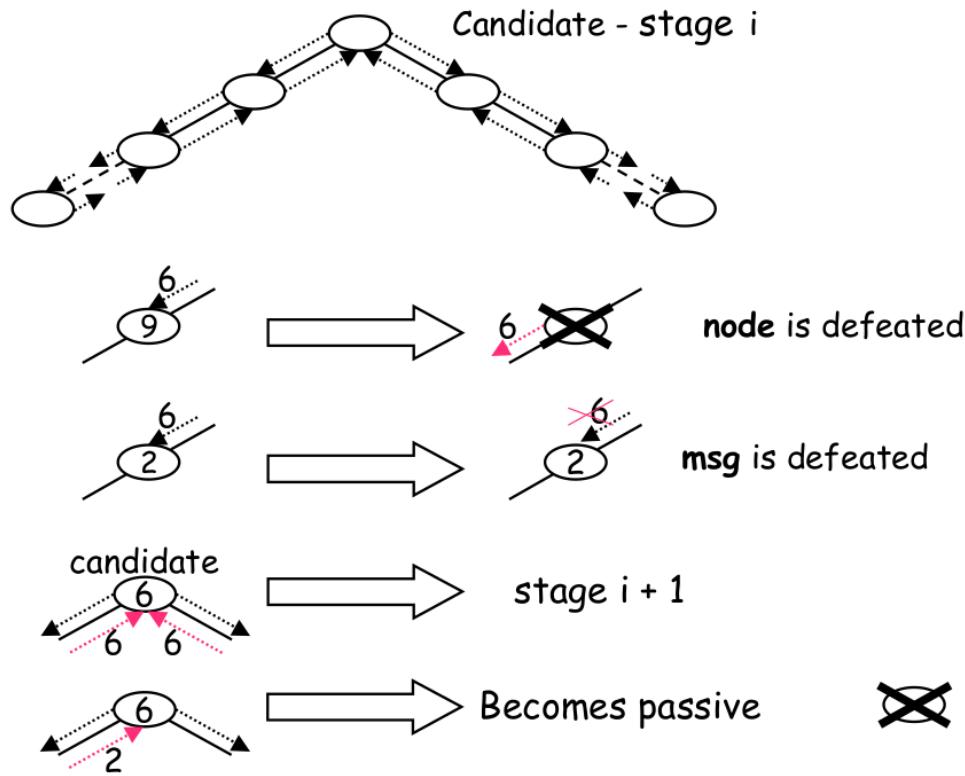
4. Un candidato che **riceve il proprio messaggio di feedback da ambo le direzioni** sopravvive ed inizia il prossimo stage

Le entità incontrate lungo il percorso leggono il msg e :

- Ogni entità  $i$  con un ID **più grande** diventa **sconfitto** (passivo)
- Un entità **sconfitta** inoltra i messaggi originati da altre entità; se il messaggio è la **notifica** di terminazione, allora termina



Vediamo un piccolo riassunto



## Correttezza e Terminazione

Se un candidato riceve uno (sinistra/destra) dei suoi messaggi dal lato opposto (destra/sinistra) lo invia, diventa il **leader** e lo notifica

Alcune proprietà :

- L'ID più piccolo percorrerà sempre la distanza massima sconfiggendo ogni entità che incontra
- La distanza aumenta monotonicamente diventando infine maggiore di  $n$
- Il Leader eventualmente riceverà il suo messaggio dalla direzione opposta

**Problema :** Che succede se un'entità riceve il messaggio da uno stage maggiore del suo attuale?

Possibile situazione cattiva : Un nodo è sconfitto da un qualche messaggio allo stage 8 e, dopo questo, riceve il suo messaggio di feedback dello stage 7

**Soluzione :** L'aproccio greedy funziona sempre!

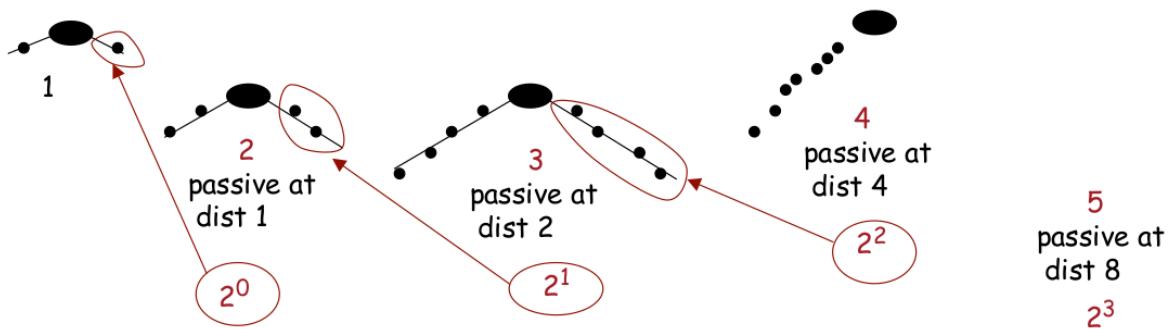
**Se a qualunque istante di tempo un nodo riceve un ID più piccolo (di qualunque stage), diventerà sconfitto e non aspetterà più per il suo messaggio di feedback**

## Message Complexity

Diamo una definizione importante :  $n_i$  numero di nodi che iniziano lo stage  $i$

☞ Lemma >

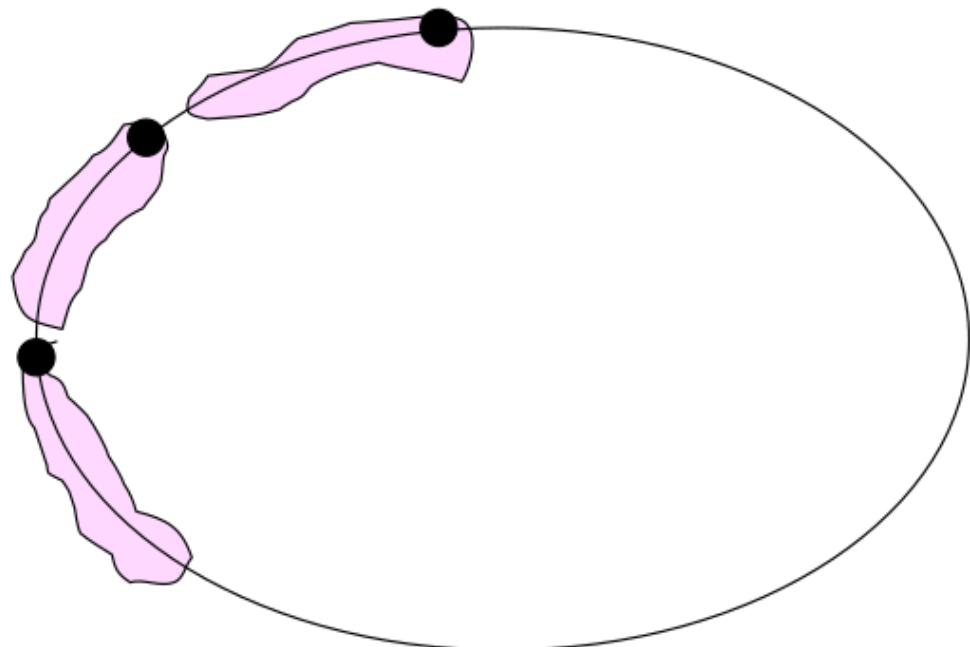
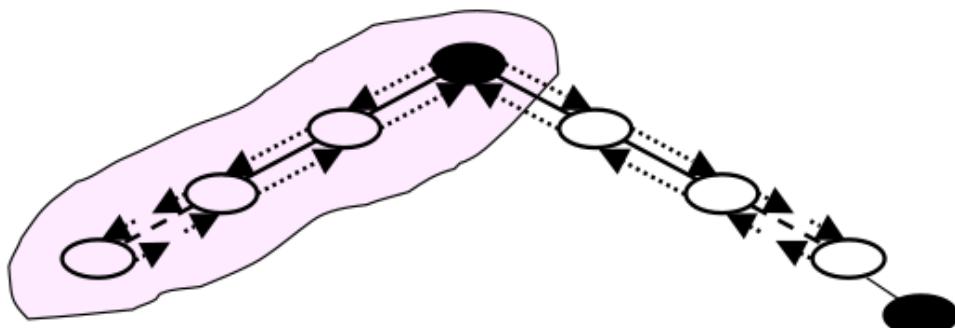
Se  $x$  inizia lo stage  $i$  (ovvero sopravvive allo stage  $i - 1$ ) l'ID di  $x$  deve essere più piccolo dell'ID dei vicini a distanza fino a  $2^{i-2}$  su ogni lato



In un gruppo di  $2^{i-2} + 1$  entità consecutive al più una può sopravvivere allo stage  $i - 1$

Quindi :

$$n_i \leq \frac{n}{2^{i-2} + 1}$$



Vediamo ora il num di messaggi.

Abbiamo due tipo di messaggi -> "Forth" e "Feedback"

### Stage $i > 1$

- "Forth" : ognuno di essi viaggerà al più  $2^{i-1}$  in entrambe le direzioni
  - Tot :  $2n_i 2^{i-1}$
- "Feedback" :
  - ogni sopravvissuto ne riceverà uno da ogni lato
    - $2n_{i+1} 2^{i-1}$
  - ogni entità che ha iniziato lo stage ma non è riuscita a sopravvivere ne riceverà uno o nessuno
    - $\leq \underbrace{(n_i - n_{i+1})}_{\text{Nodi che non superano lo stage } i} 2^{i-1}$
  - Tot :  $2n_{i+1} 2^{i-1} + (n_{i+1} - n_i) 2^{i-1}$

Quindi, mettendo le due cose insieme otteniamo

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

Per lo stage 1 la situazione è leggermente differente

### Stage 1

- Se tutti iniziano :
  - I sopravvissuti mandano  $4n_2 2^0$  messaggi -> 2 "Forth", 2 "Feedback"
  - Gli altri  $3(n - n_2) 2^0$  -> 2 "Forth", 1 "Feedback"

In totale abbiamo

$$4n_2 + 3n - 3n_2 = n_2 + 3n \quad \underbrace{<}_{\text{Ricordando } n_2 \leq \frac{n}{2^0 + 1}} \quad 4n$$

### Numero di stage totali

L'anello è attraversato completamente fintanto che  $2^{i-1}$  è maggiore/uguale a  $n$

$$2^{i-1} > n \iff i \geq \log(n) + 1$$

Quindi abbiamo in totale  $\log(n) + 1$  stage

A questo punto, la message complexity totale sarà

$$M(\text{StageTechnique}) \leq \sum_{i=1}^{\log(n)} 7n + \underbrace{O(n)}_{\text{primo stage}} = n \sum_{i=1}^{\log(n)} 7 = 7n \log(n) + O(n) \implies O(n \log(n))$$

## Congettura

### ☒ Congettura >

In anelli non direzionali, la complessità nel caso peggiore è  $n^2$ ; per avere una complessità di  $O(n \log(n))$  messaggi, la bidirezionalità è necessaria

Questa congettura però non è vera

## Stages

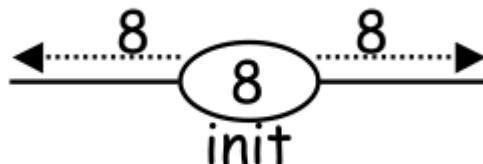
Idea base :

- Un messaggio viaggerà finchè non raggiungerà un'altro candidato.
- Un candidato riceverà un messaggio da ambo i lati

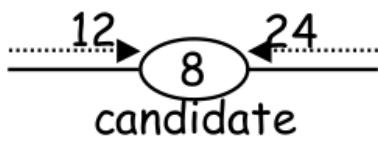
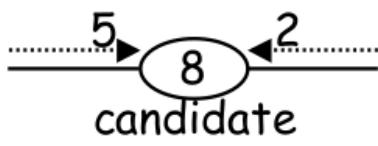
Le assunzioni sono le stesse della Stage Technique, aggiungendo il Message Ordering

Come funziona questa tecnica :

- Ogni candidato invia il suo ID in entrambe le direzioni



- Quando un candidato  $i$  riceve due messaggi  $ID_j$  (da dx) e  $ID_k$  (da sx), determina se può diventare *passivo* (ovvero lui non è il valore più piccolo), oppure se deve rimanere *candidato* (ovvero è il più piccolo)



Dopo aver ricevuto il primo messaggio, l'entità effettua l'operazione **close-port** (ovvero accoda tutti i messaggi in arrivo dopo il primo)

Dopo aver ricevuto il secondo messaggio, l'entità effettua l'operazione **re-open-port**

## Correttezza e Terminazione

L'entità con ID minimo non smetterà mai di inviare messaggi

Quando poi un'entità sa di essere Leader, invia un messaggio di Notifica che viaggerà su tutto l'anello

## Complessità - Worst Case

**Ad ogni step** : Almeno la metà delle entità diventa passiva, quindi  $n_{i+1} \leq \frac{n_i}{2}$

Quindi abbiamo che :

$$n_0 = n, n_1 \leq \frac{n}{2}, \dots, n_i \leq \frac{n}{2^i}$$

Ora,

$$\frac{n}{2^k} \leq 1 \iff k \geq \log(n)$$

Quindi abbiamo **#step** al più  $\log(n)$

Ogni entità invia o reinvia 2 messaggi, quindi :

- **#mess** :  $2n$
- **#bits** :  $2n \log(n)$

**L'ultima entità** invia  $2n$  messaggi per capire che è l'ultima entità attiva, poi  $n$  messaggi per notificare

In totale quindi :

$$M(\text{Stages}) = 2n \log(n) + 3n = O(n \log(n))$$

## Architettura Mesh

Vediamo ora il problema della Leader Election all'interno dell'architettura Mesh

Una Mesh  $M$  di dimensione  $a \times b$  ha :

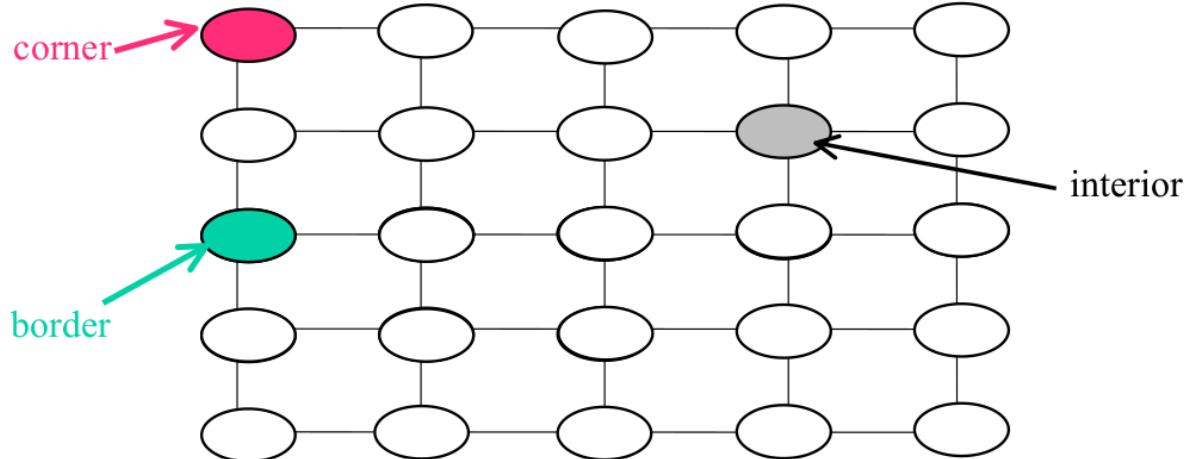
- $n = a \times b$
- $m = a(b - 1) + b(a - 1) = O(n)$

All'interno della Mesh esistono tre tipologie di nodo :

1. **Corner** : nodo che sta ai 4 angoli della Mesh

2. **Border** : nodo che sta sul perimetro della Mesh (bordo)

3. **Interior** : nodo interno alla Mesh



Alcuni fatti importanti :

1. Fatto 1 : Topologia **asimmetrica**

2. Fatto 2 : Il sottografo indotto dai Corners+Borders è un **ANELLO**

Vediamo quindi il protocollo per l'elezione nelle Mesh

## Protocollo per Leader Election

**Idea** : si elegge un Leader come uno dei 4 Corner

Questo protocollo lavora in 3 fasi :

1. Fase di **Wake Up**
2. Fase di **Elezione** (sui bordi) solo tra i Corners
3. Fase di **Notifica** (Broadcast)

Vediamo nel dettaglio

## Fase di Wake-Up + Message Complexity

**Fase di WakeUp** ( $k = \text{num. initiators}$ )

- Ogni Initiator invia un messaggio di **wake-up** a tutti i suoi vicini
- Ogni non-initiator che riceve un messaggio di **wake-up**, lo invia agli altri vicini
- In totale

$$M(\text{Wake} - \text{Up}) = 3n + k = O(n)$$

## Fase di Election

**Fase di Election**

- L'elezione avviene sul bordo dell'anello, iniziata dai Corners
- I Corners sono gli unici **idonei**
- Ogni corners  $x$  invia il suo  $ID(x)$  a **tutti**
- Ogni border inoltra ogni **nuovo** messaggio a tutti
- Ogni corners inoltra ogni nuovo messaggio agli **altri**
- Ogni interior non fa nulla

Quando un nodo può fermarsi?

Per rispondere a questa domanda dobbiamo ricordarci che stiamo in una topologia **speciale**

### Fatti cruciali

- Ogni nodo sa che si trova in una Mesh
- Ogni nodo sa che ci sono esattamente 4 messaggi che provengono dai Corners

Quindi, per rispondere alla domanda di prima, un nodo può fermarsi e decidere dopo aver ricevuto i 4 messaggi dai Corners

### Message Complexity (Fase election)

In ogni link dell'anello passano 4 messaggi, quindi in totale :  $O(a + b) = O(\sqrt{n})$

In ogni link interno, adiacente al bordo passano 4 messaggi. Anche qui il totale è  $O(a + b) = O(\sqrt{n})$

Quindi

$$M(Election) = O(\sqrt{n}) = o(n)$$

Che è ottimale perchè **sub-lineare**

### Fase di Notifica

Il Leader  $x$  invia, tramite Broadcast, il suo  $ID(x)$  su tutta la Mesh

Quindi, la message complexity è (considerando che ora stiamo nella condizione Unique Initiator) :

$$M(Flooding|RI) = O(m) = O(n)$$

**oss** : Notiamo che le fasi che costano di più sono la fase di Wake-Up e Notifica

# Lezione 9 ADRC - Deterministic Distributed Graph Coloring, Procedura BPC, Procedura KW-CRP

## Deterministic Distributed Graph Coloring

In questa sezione, studieremo vari metodi per risolvere il problema del coloramento su sistemi distribuiti.

Più nello specifico, vogliamo cercare di ottenere una colorazione del sistema come se stessimo risolvendo il problema COL su grafi centralizzati.

Prima di vedere le due procedure fondamentali, diamo la definizione formale del problema.

### Problema Distribuiti Coloring (DCP)

- Modello del sistema
  - Il sistema distribuito è formalizzato come un grafo etichettato  $G(V, E, Id)$ , con  $\Delta$  il grado max. del grafo  $G$
  - Ci mettiamo nel modello *LOCAL*, sotto le assunzioni Standard, e con l'utilizzo di un clock globale che scandisce il tempo in round sincroni  $t = 1, 2, \dots$  (notiamo quindi che stiamo nel sistema **sincrono** e non più asincrono)
- Definizione del problema DCP
  - **Configurazione iniziale** : un coloramento (iniziale) *legale*  $C : V \rightarrow [n]$  t. c.  $C := ID$ , dove ogni nodo  $v$  conosce il suo coloramento  $C(v)$
  - **Configurazione finale** : un coloramento legale  $\hat{C} : V \rightarrow [\Delta + 1]$ , dove ogni nodo  $v$  conosce il suo coloramento finale  $\hat{C}$

Ricordiamo che per coloramento legale intendiamo un coloramento dove

$$\forall x, y \in V, (x, y) \in E \mid C(x) \neq C(y)$$

#### ☞ Teorema >

Ogni grafo  $G(V, E)$  con grado massimo  $\Delta$  **ammette un coloramento legale** che usa al più  $\Delta + 1$  colori

Il nostro goal è quindi quello di trovare un  $(\Delta + 1)$ -coloramento nel modo distribuito.

#### ⓘ Remark + Oss >

Piccolo remark sul coloramento, si ha che  $3 - COL \in NPC$

Osservazione : Non è detto che il  $(\Delta + 1) - COL$  sia quello ottimale, infatti possono esserci topologie di sistema distribuito tale che il coloramento ottimale sia ad es. il  $2 - COL$

Vediamo ora la prima procedura per questo problema

## The Basic Color Reduction Procedure (BCP)

**BCP Task su G :**

- **Configurazione iniziale** : un coloramento (iniziale)  $C : V \rightarrow [a]$  t. c.  $a > \Delta + 1$ , dove ogni nodo  $v$  conosce il suo coloramento  $C(v)$
- **Configurazione finale** : un  $(\Delta + 1)$ -coloramento  $\hat{C} : V \rightarrow [\Delta + 1]$ , dove ogni nodo  $v$  conosce il suo coloramento finale  $\hat{C}$

La procedura è la seguente :

- for each  $k = a, a - 1, \dots, \Delta + 2$  do (in modo sequenziale)
  - each node  $v$  do in parallelo
    - $v$  invia il suo colore  $C(v)$  a tutto  $N(v)$  ;  $v$  riceve da tutti i suoi vicini i colori da loro usati  $C(N(v))$
    - (\*\*\*) If  $C(v) = k$  then
      - $v$  sceglie un qualunque  $\hat{j} \in \underbrace{[k - 1] - N(C(v))}_{\text{Esiste sempre per (*)}}$  e imposta  $C(v) = \hat{j}$
      - (sostanzialmente sceglie un nuovo colore dalla tavolozza di colori disponibili, togliendo però i colori usati dal suo vicinato)
  - Ogni nodo  $v$  imposta il suo colore finale  $\hat{C}(v) = C(v)$  e si ferma

(\*) : questo vale perchè  $k \geq \Delta + 2 \wedge deg(v) \leq \Delta$

## BCP : Analisi I (Correttezza)

- **Fatto 1** : Ad ogni fase  $k$ , per ogni nodo  $v$ , il colore  $\hat{j}$  esiste sempre.
- **Fatto 2** : Alla fine di ogni fase  $k > \Delta + 1$ , il coloramento  $C : V \rightarrow [a - k]$  è legale
- **Corollario** : BCP ritorna un  $(\Delta + 1)$ -coloramento  $\hat{C} : V \rightarrow [\Delta + 1]$  legale.

Dimostriamo :

Il fatto 1 è trivia, il fatto 2 è conseguenza del fatto che i nodi con colore  $k$  non sono **mai adiacenti** per definizione di coloramento e per lo step (\*\*) di BCP

## BCP : Analisi 2 (Time/Message Complexity)

## Vediamo la **time complexity**

Abbiamo che il numero di fasi che la procedura BCP eseguirà sono  $\Theta(n - \Delta)$ , ma  $1 \leq \Delta \leq n - 1$ , di conseguenza il worst-case si ha quando  $\Delta = O(1) \implies$  numero di fasi risulta essere  $\Theta(n)$  (non efficiente)

## Vediamo invece la **message complexity**

Per ogni arco passano 2 messaggi  $\rightarrow 2m$

La procedura lavora in  $\Theta(n - \Delta)$  round, quindi otteniamo che :

$$M(BCP) = \Theta(m(n - \Delta))$$

In generale, vale il seguente teorema :

### Teorema generale >

Su un grafo  $G(V, E, Id)$ , con coloramento iniziale  $C := ID$  (e con  $a = n$ ), la procedura BCP converge entro  $O(n - \Delta)$  round a un  $(\Delta + 1)$ -coloramento, e ha message complexity pari a  $O((n - \Delta)m)$

Vediamo adesso l'altra procedura, quella che sfrutta la parellizzazione in modo molto più efficiente

## Khun-Wattenhofen Color Reduction Procedure (KW-CRP)

Configurazione iniziale e finale come per il task *BCP*

**KW-CRP :**

- Dato un coloramento  $C : V \rightarrow [a]$ , con  $a > \Delta + 1$ , si partiziona l'insieme  $V$  :
  - impostiamo  $k = \frac{a}{\Delta+1}$ , e ogni nodo calcola la sua partizione  $V_i$  nel seguente modo
 
$$V_i = \{v \in V : (i - 1) \cdot (\Delta + 1) + 1 \leq C(v) \leq i \cdot (\Delta + 1)\}, i = 1, \dots, k$$
  - ricordiamo che  $V = \{V_1, V_2, \dots, V_k\}$

**Fatto 3 :** Il coloramento  $C : V \rightarrow [a]$  induce un  $(2(\Delta + 1))$ -coloramento, chiamato  $F_{12}$  sul **sottografo indotto**  $G(V_1 \cup V_2, E)$ , infatti

$$F_{12} : V_1 \cup V_2 \rightarrow [2(\Delta + 1)]$$

**oss :** Il coloramento  $C : V \rightarrow [a]$  è un  $a$ -coloramento su *ogni* sottografo indotto  $G$

## KW-CRP : Ingredienti Chiave

**Fatto 4 :** Definiamo  $F_{i(i+1)} : V_i \cup V_{i+1} \rightarrow [2(\Delta + 1)]$  per ogni  $i = 1, \dots, k - 1$  come nel fatto 3.

Allora vale :

1.  $F_{i(i+1)} : V_i \cup V_{i+1} \rightarrow [2(\Delta + 1)]$  è un  $[2(\Delta + 1)]$ -coloramento per il sottografo  $G(V_i \cup V_{i+1}, E)$
2. L'insieme dei colori usati nei vari  $F_{i(i+1)}$  sono mutualmente disgiunti

**KW-CRP Idea Chiave (Fase):** Applicare BCP, **in parallelo**, su ogni sottografo indotto  $G(V_i \cup V_{i+1}, E)$ , partendo dal  $[2(\Delta + 1)]$ -coloramento  $F_{i(i+1)}$

- la procedura BCP trasformerà ogni  $[2(\Delta + 1)]$ -coloramento iniziale  $F_{i(i+1)}$ , in un  $[(\Delta + 1)]$ -coloramento  $F_{i(i+1)}$  per il sottografo indotto  $G(V_i \cup V_{i+1}, E)$
- Il numero di round paralleli di una Fase è pari a  $\Delta$

**Fatto 5 :**

1. Dopo **una** applicazione parallela della fase BCP, la funzione globale  $\langle F_{12}, \dots, F_{i(i+1)}, \dots, F_{(k-1)k} \rangle$  diventa un  $[\frac{a}{2}]$ -coloramento legale per il grafo  $G$ .  
(Conseguenza di Fatto 4 e del fatto che gli archi che collegano due classi  $V_i, V_j$ , detti **Ponti**, non corrompono il coloramento fra queste due classi)
2. Dopo  $t$  applicazioni parallele della fase BCP, la funzione globale  $\langle F_{12}, \dots, F_{i(i+1)}, \dots, F_{(k-1)k} \rangle$  diventa un  $[\frac{a}{2^t}]$ -coloramento per il grafo  $G$

## KW-CRP : Time Complexity

### ☞ Teorema >

Partendo da un coloramento iniziale (es.  $a = n$ ), dopo  $t = \log(\frac{n}{\Delta})$  fasi parallele, la funzione globale  $\langle F_{12}, \dots, F_{i(i+1)}, \dots, F_{(k-1)k} \rangle$  sarà trasformata in un  $(\Delta + 1)$ -coloramento per il grafo  $G$ .

Essendo che ogni fase impiega tempo  $\Delta$ , si avrà che la time complexity generale sarà  $O(\Delta \cdot \log(\frac{n}{\Delta}))$

**dim**

Questo vale perchè (Fatto 5,p.2) dopo  $t$  applicazioni avremo un  $[\frac{a}{2^t}]$ -coloramento per il grafo  $G$

Ora, vediamo i calcoli :

$$\frac{a}{2^t} \leq \Delta + 1 \implies 2^t \geq \frac{a}{\Delta} \xrightarrow{\text{minimo } t} t = \Theta\left(\log\left(\frac{n}{\Delta}\right)\right)$$

Ripetendo il tutto per  $\Delta$  volte otteniamo

$$Time(\text{KW-CRP}) = \Theta\left(\Delta \log\left(\frac{n}{\Delta}\right)\right)$$

## KW-CRP : Message Complexity

Su ogni arco passano 2 messaggi, quindi il num. di messaggi scambiati sarà  $\Theta(2m)$   
Ripetendo  $\Delta \log\left(\frac{n}{\Delta}\right)$  volte otteniamo che :

$$M(\text{KW-CRP}) = \Theta\left(m \left[ \Delta \log\left(\frac{n}{\Delta}\right) \right]\right)$$

# Lezione 10 ADRC - Algoritmi Randomizzati per Coloring Distribuito, Procedura Rand-2Delta

## Primo algoritmo randomizzato per CD

### Phase Technique

Il paradigma della **Basic Phase Technique**

While  $V \neq \emptyset$  do : (**Ogni esecuzione è una fase**)

1. Ogni nodo sopravvissuto  $v$  esegue in parallelo :
  1. Sceglie u.a.r un colore  $C(v)$  da un qualunque insieme  $Z$  e informa tutto il suo vicinato  $N(v)$
  2. In base al colore ricevuto dal suo vicinato,  $C(N(v))$ , imposta
    1. (i)  $C(v)$  come "sbagliato"
    2. (ii)  $C(v)$  come "okay"
2. Se succede (i),  $v$  **sopravvive** e continua a cercare il suo colore nella prossima fase
3. Se succede (ii),  $v$  **ottiene** il suo colore finale  $C(v)$ , informa  $N(v)$  e viene rimosso da  $G$ , ottenendo così un nuovo grafo  $G'(V', E')$ , con

$$\begin{aligned} V' &= V \setminus \{v\} \\ E' &= E \setminus \{(u, v) \in E, v \in V\} \end{aligned}$$

Dopo aver definito la Phase Technique, passiamo a definire il protocollo che sfrutterà questa tecnica, chiamato Rand-2Δ

## Protocollo Rand-2Delta

**Algorithm 18** Procedure Rand-2Delta( $G$ )

An algorithm for each vertex  $v \in V$ .

Initially  $T_v = \emptyset$ ,  $F_v = \emptyset$ , for each  $v \in V$ .

*/\*  $T_v$  is the set of temporary colors selected by neighbors of  $v$ . \*/*

*/\*  $F_v$  is the set of final colors selected by neighbors of  $v$ . \*/*

```

1: for each round do
2:    $T_v = \emptyset$ 
3:    $c_v :=$  draw a color from  $[2\Delta]$  u.a.r., independently of other vertices
4:   send the color  $c_v$  to all neighbors
5:   for each received color  $c_u$  from a neighbor  $u$  do
6:      $T_v := T_v \cup \{c_u\}$ 
7:   end for
8:   if  $c_v \notin T_v \cup F_v$  then
9:     send the message "final  $c_v$ " to all neighbors
10:    select  $c_v$  as the final color  $\varphi_v$  of  $v$  and terminate
11:   else
12:     for each received message "final  $c_u$ " from a neighbor  $u$  do
13:        $F_v := F_v \cup \{c_u\}$ 
14:     end for
15:     discard  $c_v$  and continue to the next round
16:   end if
17: end for
```

**Fatto 1 :** Se *tutti* i nodi terminano (es.  $V = 0$ ), allora la procedura Rand- $2\Delta$  calcola un (legale)  $(2\Delta)$ -coloramento per il grafo  $G$

**dim :**

Dagli Step 6 – 10, quando  $v$  ottiene il suo colore finale  $C(v)$  allora è considerato sicuro. Tutti i suoi vicini sono o non ancora  $F$ -colorati, oppure sono colorati con altri colori. Questo perchè  $C(v)$  è sempre scelto nell'insieme  $[2\Delta] \setminus (F_v \cup T_v)$

## Rand-2Delta : Analisi

**Fatto 2 :** Dopo  $O(\log(n))$  Fasi, con alta probabilità *tutti* i nodi termineranno.

**dim**

Fissiamo un nodo  $v \in V$  e una fase  $t > 0$

Abbiamo che  $|F_v \cup T_v| \leq \Delta$ , in quanto ogni vicino del nodo  $v$  contribuisce con al più un colore a tale insieme, e il grafo  $G$  è  $\Delta$ -regolare (cioè  $|N(v)| = \Delta$ )

Di conseguenza, abbiamo che

$$\Pr[v \text{ termina alla fase } t | v \text{ non è terminato prima}] \geq \frac{2\Delta \setminus (F_v \cup T_v)}{2\Delta} = \frac{1}{2}$$

Applicando la regola della catena (non importa il passato: si ha sempre il 50% di chance),

vale che

$$\Pr[\mathcal{E}_v = v \text{ non termina dopo } t \text{ Fasi}] \leq \left(\frac{1}{2}\right)^t$$

A questo punto, usando lo **Union Bound** su tutti gli  $n$  eventi "cattivi"  $\mathcal{E}_v$ , otteniamo che :

$$\Pr[\mathcal{E} = \exists v \in V : v \text{ non termina dopo } t \text{ Fasi}] \leq n \left(\frac{1}{2}\right)^t$$

Quindi, per  $t \geq (c+1) \log(n)$  abbiamo che

$$\Pr[\mathcal{E}] \leq \frac{1}{n^{c-1}} \implies \Pr[\hat{\mathcal{E}}] = 1 - \Pr[\mathcal{E}] \leq 1 - \frac{1}{n^c}$$

E quindi abbiamo trovato che, con alta probabilità, ogni nodo termina entro  $O(\log(n))$  rounds

# Lezione 11 ADRC - Ancora su CD, Algoritmo Rand-Delta+, Random Election Leader, Maximal Indipendent Set Distribuito

## Un'altro algoritmo randomizzato per CD

Nella lezione scorsa avevamo introdotto un primo algoritmo randomizzato per il problema del Coloring Distribuito, ottenendo un  $(2\Delta)$ -coloramento.

Ora vediamo il secondo algoritmo distribuito, che da  $2\Delta$  ci farà raggiungere il  $(\Delta + 1)$ -coloramento.

## Procedura Rand-Delta+

In questa procedura facciamo due **cambiamenti chiave**

In ogni fase :

1. Ogni nodo sopravvisuto  $v$  prima sceglie u.a.r un bit  $c(v) \in \{0, 1\}$ 
  1. Se  $c(v) = 0$  allora  $v$  viene **sconfitto** e salta alla prossima fase
  2. Altrimenti, il nodo  $v$  sceglie un colore  $C(v)$  u.a.r dalla palette  $[\Delta + 1] \setminus F_v$

A questo punto, il protocollo procede come Rand- $2\Delta$

---

### Algorithm 19 Procedure Rand-Delta-Plus1( $G$ )

---

An algorithm for each vertex  $v \in V$ .

Initially  $T_v = \emptyset$ ,  $F_v = \emptyset$ , for each  $v \in V$ .

```

1: for each round do
2:    $T_v := \emptyset$ 
3:    $c_v :=$  draw u.a.r. a bit from  $\{0, 1\}$ 
4:   if  $c_v = 0$  then
5:     discard  $c_v$  and continue to the next round
6:   else
7:      $c_v :=$  draw u.a.r. a color from  $[\Delta + 1] \setminus F_v$ , independently of other vertices
8:     send the color  $c_v$  to all neighbors
9:     for each received color  $c_u$  from a neighbor  $u$  do
10:       $T_v := T_v \cup \{c_u\}$ 
11:    end for
12:    if  $c_v \notin T_v \cup F_v$  then
13:      send the message “final  $c_v$ ” to all neighbors
14:      select  $c_v$  as the final color  $\varphi_v$  of  $v$  and terminate
15:    else
16:      for each received meassage “final  $c_u$ ” from a neighbor  $u$  do
17:         $F_v := F_v \cup \{c_u\}$ 
18:      end for
19:      discard  $c_v$  and continue to the next round
20:    end if
21:  end if
22: end for

```

---

**Fatto 3 :** Se *tutti* i nodi terminano (es.  $V = 0$ ) , allora il protocollo Rand- $\Delta^+$  calcola un (legale)  $(\Delta + 1)$ -coloramento per il grafo  $G$

**dim :** identica a quella per il protocollo Rand- $2\Delta$

## Rand-Delta+ : Analisi

**Fatto 4 :** Dopo  $O(\log(n))$  fasi, con alta probabilità, *tutti* i nodi terminano

**dim :**

Fissiamo  $v$  e una fase  $t$

Per un  $u \in N(v)$  fissato, abbiamo che

$$\Pr[c(u) = c(v)] \leq \frac{1}{2(\Delta + 1 \setminus |F_v|)} \quad (1)$$

Usando lo Union Bound su ***tutti i vicini sopravvissuti***  $u \in N(v)$  ***che hanno scelto***  $b = 1$ , dall'equazione (1) si ha che

$$\Pr[\mathcal{E}_v = \exists u \in N(v) : c(u) = c(v)] \leq \frac{\frac{\text{num. vicini attivi di } v}{(\Delta + 1 \setminus |F_v|)}}{2(\Delta + 1 \setminus |F_v|)} \leq \frac{1}{2} \quad (2)$$

Quindi, si ha che

$$\Pr[c(v) > 0 \wedge \neg \mathcal{E}_v] \geq \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

Da qui in poi, si prosegue come la dimostrazione del Fatto 2

## Random Leader Election

Spostiamoci dal problema del Coloring Distribuito, e torniamo per un secondo al vecchio problema della Leader Election.

Ricordando che nella Leader Election avevamo mostrato che, senza l'assuzione di Unique Identifier, era impossibile risolvere problema in modo **deterministico**

Qui studiamo un'approccio probabilistico che risolve il problema della Leader Election, nella situazione in cui tutti i nodi sono **anonimi**

## RLE in Grafi non etichettati

L'idea del protocollo è la seguente : ogni nodo sceglie u.a.r un numero in  $[m]$ , tale numero diventerà l'ID del nodo.

Dopo questa fase di *labeling*, si procede esattamente come la controparte deterministica.

Descriviamo ora, e in modo formale il protocollo randomizzato, su un modello sincrono :

### Randomize Protocol RLE $G(V, E), |V| = n, |E| = m$

- Fase 0 : Wake-Up : tutti i nodi vengono attivati
- Fase 1 : Ogni nodo  $v$  sceglie u.a.r un intero  $j_v \in [m]$
- Fase 2 : Ogni nodo esegue un protocollo deterministico fissato per il  $LE$  assumendo il labeling dei nodi come  $\{j_v : v \in V\}$

## Analisi del Protocollo

Analizziamo la correttezza del protocollo *RLE*.

Per la prima volta, non possiamo dimostrare che il protocollo è sempre corretto, ovvero converge sempre a uno stato finale.

Infatti, si può facilmente verificare che, nel possibile scenario in cui ogni nodo dopo la fase 2, sceglie la stessa etichetta  $m$ , il protocollo fallisce!

Chiaramente, questo evento è estremamente improbabile, ma è possibile!

Il nostro primo passo è quello di ricavare una condizione specifica, cioè un evento, che sia sufficiente a sostenere che, assumendo che quell'evento si verifichi, il protocollo funzioni correttamente. Nel nostro caso, osserviamo che una condizione sufficiente (non necessaria) è il seguente evento :

$$\mathbb{B} = \text{"non esiste nessuna coppia di nodi differenti } v, w \in V, v \neq w : j_v = j_w \text{"}$$

Infatti, come già osservato, se tutte le etichette  $j_v$  sono mutualmente differenti, allora il protocollo nella fase 3 funziona come un protocollo deterministico standard di Leader Election, sotto l'ipotesi *UniqueIdentifier*(UI)

Possiamo vedere la Fase 2 del protocollo come un classico processo *Balls-into-Bins* in cui ci sono  $n$  palline (cioè le scelte casuali  $j_v$ ) che vengono lanciate indipendentemente e u.a.r in  $m$  bins (i possibili valori assunti dalle etichette).

La domanda che ci interessa è quanto è grande la probabilità che due palline diverse cadano nello stesso contenitore.

A tal fine, consideriamo un bin fisso  $i \in [m]$ .

Allora la probabilità dell'evento

$$\mathbb{B}_i = \text{"Almeno due palle entrano nel bin i-esimo"}$$

è

$$\begin{aligned}
Pr(\mathbb{B}_i) &= Pr\left(\bigcup_{k=2}^n \{\text{esattamente } k \text{ palle finiscono nel bin } i\text{-esimo}\}\right) \\
&= \sum_{k=2}^n \binom{n}{k} \left(\frac{1}{m}\right)^k \underbrace{\left(1 - \frac{1}{m}\right)^{n-k}}_{\leq 1} \text{ (eventi disgiunti)} \\
&\leq \sum_{k=2}^n \binom{n}{k} \left(\frac{1}{m}\right)^k \\
&\leq \sum_{k=2}^n \left(\frac{en}{k}\right)^k \left(\frac{1}{m}\right)^k \text{ (Appx di Stirling)} \\
&\leq \left(\frac{en}{2m}\right)^2 + n \left(\frac{en}{3m}\right)^3 = O\left(\frac{n^2}{m^2} + \frac{n^4}{m^3}\right)
\end{aligned}$$

Osserviamo che il bound di cui sopra si riferisce a un bin  $i$ -esimo fissato.

Il protocollo va fallisce quando **almeno** uno dei bin ottiene l'evento  $\mathbb{B}_i$

Sia ora l'evento  $\mathbb{B}$  l'unione degli eventi  $\mathbb{B}_i$  al variare  $i \in [m]$ , quindi applicando lo Union Bound abbiamo che :

$$\begin{aligned}
Pr(\mathbb{B}) &\leq Pr\left(\bigcup_{i=1}^m \mathbb{B}_i\right) \\
&\leq \sum_{i=1}^m Pr(\mathbb{B}_i) \\
&\leq m \cdot Pr(\mathbb{B}_i) \\
&= m \cdot O\left(\frac{n^2}{m^2} + \frac{n^4}{m^3}\right) \\
&= O\left(\frac{n^2}{m} + \frac{n^4}{m^2}\right)
\end{aligned}$$

A questo punto, fissando il parametro  $m$  come  $m \geq n^3$  otteniamo che

$$Pr(\mathbb{B}) = O\left(\frac{1}{n}\right)$$

Abbiamo quindi dimostrato che il protocollo RLE con input  $(G, n, n^3 = m)$  esegue una corretta Leader Election su un anello di dimensione  $n$ , w.h.p ■

## Maximal Indipendent Set Distribuito

- **System Model e Restrizioni**
  - Grafo etichettato  $G(V, E), ID$ , con  $|V| = n$  nodi e  $|E| = m$  archi
  - Restrizioni Standard, Modello Local e Attivazioni Parallelle Sincrone  $t = 1, 2, \dots$

### Def (MIS)

- *Configurazione Iniziale* : un'etichettamento (iniziale)  $I : V \rightarrow \{0, 1\}$  tale che  $I(v) = 0, \forall v \in V$
- *Configurazione Finale* : un'etichettamento  $I : V \rightarrow \{0, 1\}$  tale che l'insieme  $M = \{v \in V : I(v) = 1\}$  forma un Indipendent Set **Massimale**

Remark Massimale :  $I$  indipendent set massimale significa che  $\forall z \in V \setminus I \implies I \cup \{z\}$  non è più IS

## Procedura Base per MIS

Qualche notazione :

- $N(v)^+ = N(v) \cup \{v\}$
- $N^+(S) = S \cup (\bigcup_{s \in S} N(s))$

**MIS Task** su grafo  $G$  :

- Imposta  $M = \emptyset$
- While  $V \neq \emptyset$ 
  - i. Calcola un indipendent set  $S \subseteq V$  per  $G$  (**Step (a)**)
  - ii.  $M = M \cup S$
  - iii.  $V = V \setminus N^+(S); E = E \setminus E(N^+(S))$  dove  $E(N^+(S)) = \{(u, v) : v \in N^+(S)\}$
- Per ogni nodo  $v \in V$ , imposta  $I(v) = 1 \iff v \in M$

### ☞ Teorema >

Sotto le ipotesi definite in precedenza, la procedura MIS-P ritorna sempre un MIS per il grafo  $G$

**dim**

- (a) ogni coppia di nodi in  $M$  non può essere collegata da un'arco, a causa dello step iii definito dall'algoritmo
- (b) solo i nodi (e gli archi) che sono in  $N^+(S)$  sono rimossi ad ogni round. I nodi che stanno in  $N^+(S) \setminus S$  sono esattamente quelli che **NON** possono essere inseriti in  $M$  nei prossimi rounds, e quindi  $M$  è **massimale**

## MIS-P : Analisi e Problemi

**Problema 1** : Nello step (a) dobbiamo calcolare un Indipendent Set  $S$  per  $G$ , ma la domanda è

- come possiamo eseguire questo task nel mondo distribuito? I nodi devono sincronizzare le loro scelte

## Primo tentativo (inefficiente) : **Facciamo Leader Election sui Vicinati**

- Ogni nodo  $v$  tira una "moneta" (sceglie un random bit  $b(v)$ )
  - Ogni nodo  $v$  che (i) ottiene 1 mentre (ii) **tutti** i suoi vicini ottengono 0 (in questo caso diremo che  $v$  ha "vinto la lotteria") sarà inserito in  $S$
  - Questo evento può essere controllato localmente con una comunicazione one-round
  - **Problema Tecnico Fondamentale** : Qual'è la probabilità che per un nodo fissato  $v \in N^+(v)$  vinca la lotteria? Purtroppo molto piccola, infatti la probabilità è

$$\frac{1}{2} \cdot \frac{1}{2^{|N^+(v)|}} \simeq \frac{1}{2^{|N^+(v)|}}$$

Questo implica un tempo atteso **esponenziale**

Vediamo ora la procedura per migliorare questa situazione

## Il Protocollo di Luby (LP)

Il protocollo è il seguente, e sfrutta l'idea che ogni nodo sceglie u.a.r un valore di priorità

**MIS Task** su  $G$  ; Parametro  $N \in \mathbb{N}$  (verrà definito dopo)

- Imposta  $M = \emptyset$
- While  $V \neq \emptyset$ 
  - Ogni nodo  $v$  sceglie u.a.r un valore  $r(v) \in [N]$
  - Sia  $S = \{v \in V : r(v) = \max\{r(w) : w \in N(v)\}\}$
  - Sia  $M = M \cup S$
  - $V = V \setminus N^+(S); E = E \setminus E(N^+(S))$  dove  $E(N^+(S)) = \{(u, v) : v \in N^+(S)\}$
- Per ogni nodo  $v \in V$ , imposta  $I(v) = 1 \iff v \in M$

## LP : Analisi

Impostiamo il parametro di LP come  $N = \Theta(n^3)$

Perchè lo impostiamo a questo valore? La risposta ce la da il seguente lemma :

**Lemma** : Ad ogni round  $t \geq 1$  abbiamo che le priorità  $r(v)$  sono **tutte mutualmente indipendenti** (e stocasticamente indipendenti), e quindi con alta probabilità vale che non esiste nessuna coppia di nodi che ha lo stesso valore di priorità

**dim**

La probabilità che due nodi vicini abbiano lo stesso valore di priorità è

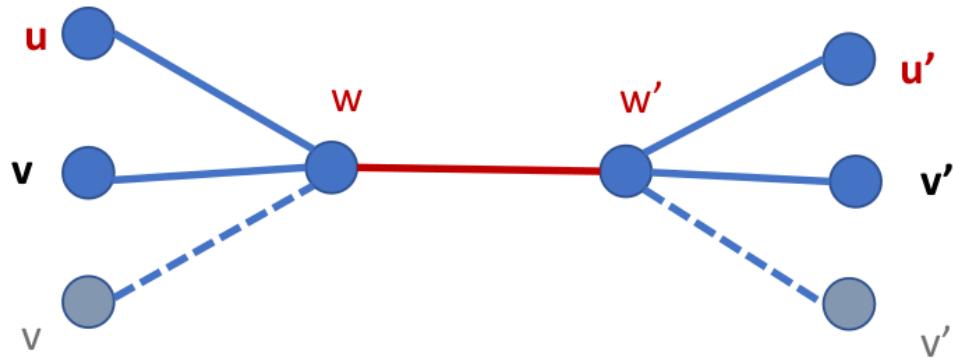
$$Pr[\exists (u, v) \in E : r(u) = r(v)] = Pr\left(\bigcup_{(u,v) \in E} r(v) = r(u)\right) \leq \sum_{(u,v) \in E} Pr[r(v) = r(u)] = \frac{|E|}{R}$$

Ora, nel caso peggiore  $|E| = m = n^2$ . Di conseguenza, se scegliamo  $R = n^c = n^{3+\varepsilon}$  otteniamo che la prob. descritta sopra è  $\simeq \frac{1}{n^c}$ , e quindi abbiamo ottenuto che con alta probabilità il Lemma 1 vale.

Ora, per proseguire consideriamo che il Lemma 1 vale con probabilità 1.

Vogliamo ora vedere quanti archi vengono rimossi ad ogni round del protocollo.

Consideriamo il seguente esempio :



**Goal** : Calcolare il numero (atteso)  $e(t)$  di archi che sono **rimossi** da  $E$  ad ogni round di LP.  
Per fare ciò, introduciamo l'**azione** : "nodo  $u$  \*salva  $w$ "

Per ogni inserimento  $v \rightarrow S$ , sappiamo che  $r(v)$  è la priorità *maggior*e in  $N^+(v)$ . Quindi, come possiamo vedere dalla figura sopra :

Se  $u, v \rightarrow S$ , allora  $r(v)$  è la *maggior*e priorità in  $N(u)$  e  $r(u)$  è la *maggior*e priorità in  $N(v)$

Diamo ora una definizione :

### ☒ Definizione >

Sia il nodo  $u$  tale che

$$r(u) = \max\{r(v) : v \in N(u)\} \quad (1)$$

Allora diciamo che l'arco  $(w, w')$  viene *rimosso* perché **il nodo  $u$  salva  $w$**

A questo punto :

**Claim I** : Dall'equazione (1) otteniamo che  $r(u) = \max\{r(v) : v \in N^+(u) \cup N^+(w)\} \rightarrow$  Evento  $B$

A questo punto possiamo affermare il seguente lemma

**Lemma 1** : Ad ogni round del protocollo LP, per ogni nodo  $u \rightarrow S$ , si ha che :

$$\Pr[B] \geq \frac{1}{d(u) + d(v)} \quad (2)$$

**Dim :** Conseguenza diretta dell'equazione (1) e del fatto che tutte le priorità sono mutualmente indipendenti

**Oss :** Nell'equazione (2) abbiamo un uguaglianza quando  $N(u) \cap N(w) = \emptyset$

Ora, definiamo una v.a  $X$  in questo modo :

### ☒ Variabile Aleatoria X Binaria >

Per un  $u \rightarrow S$ , per  $w \in N(u)$ , definiamo la v.a binaria

$$X(u \rightarrow w) = \begin{cases} 1 & u \text{ salva } w \\ 0 & \text{altrimenti} \end{cases}$$

Ora, dall'equazione (2) otteniamo che

$$\Pr[X(u \rightarrow w) = 1] \geq \frac{1}{d(u) + d(w)} \quad (3)$$

Ora siamo pronti a determinare il valore atteso di archi rimossi ad ogni round  $t$

**Lemma 2 :** Ad ogni round  $t$ , il numero  $Y$  di archi **rimossi** soddisfa l'equazione :

$$\begin{aligned} \mathbb{E}(Y) &\geq \frac{1}{2} \cdot \sum_{u,w \in E} \Pr[X(u \rightarrow w) = 1] \cdot d(w) \text{ (dato che se } u \rightarrow w \text{ allora } u \text{ salva tutto } N(w)) \\ &= \frac{1}{2} \left[ \sum_{u,w \in E} \Pr[X(u \rightarrow w) = 1] \cdot d(w) + \sum_{u,w \in E} \Pr[X(w \rightarrow u) = 1] \cdot d(u) \right] \\ &\geq \frac{1}{2} \cdot \sum_{u,w \in E} \left[ \frac{d(w)}{d(u) + d(w)} + \frac{d(w)}{d(u) + d(w)} \right] \\ &= \frac{|E|}{2} \end{aligned}$$

**Oss :** Perchè abbiamo il fattore  $\frac{1}{2}$ ? Perchè, come possiamo vedere dalla figura precedente, ci **possono essere al più** 2 leader  $u$  e  $u'$  che possono *salvare* lo **stesso** arco  $(w, w')$ . Quindi ogni "salvataggio" dei leader deve essere ridotto di un fattore 50%

## LP : Tempo di convergenza

### ☒ Teorema 1 [Tempo di convergenza atteso del protocollo LP] >

Sotto le restrizioni standard, il protocollo LP con input il grafo  $G$  ha come tempo atteso di **convergenza** pari a  $O(\log(n))$

## Dimostrazione

Definiamo

$$m(t) = |\overbrace{E(t)}^{\text{num. archi a tempo t}}|$$

Allora il Lemma 2 implica che, per ogni  $t \geq 0$  :  $\mathbb{E}[m(t+1)] \leq \frac{m(t)}{2}$

Usando l'**equazione di Markov**, otteniamo che

$$\Pr \left[ m(t+1) \geq \frac{3}{2} \frac{m(t)}{2} \right] \leq \frac{\binom{\frac{m(t)}{2}}{2}}{\frac{3}{2} \frac{m(t)}{2}} = \frac{2}{3} \quad (5)$$

Definiamo ora la v.a binaria  $Z(t) = \begin{cases} 1 & m(t) \leq \frac{3}{4}m(t-1) \\ 0 & \text{altrimenti} \end{cases}$  e sia  $Z(T) = \sum_{t=1}^T Z(t)$

Dall'equazione (5) otteniamo che

$$\mathbb{E}[Z(T)] = \sum_{t=1}^T \mathbb{E}[Z(t)] \geq \frac{2}{3}T$$

Quindi, per  $T = c \cdot \log(|E(0)|)$ , otteniamo che  $E(T) = \emptyset$ , per una costante  $c$  sufficientemente grande. (oss.  $E(0)$  è il num. di archi a tempo 0, quindi è pari a  $m \approx n$ )

Di conseguenza, quando  $E(T) = \emptyset$  il protocollo termina globalmente, ritornando in output il MIS. ■

# Lezione 12 ADRC - Ultimo argomento modello LOCAL -> Averaging Dynamics

## Concetti base di algebra lineare

Norma finita standard di un vettore  $\hat{x} = \langle x(1), \dots, x(n) \rangle \in \mathbb{R}^n$  :

$$\text{per ogni fissato } p \in \mathcal{N} : \|\hat{x}\|_p = \left( \sum_j |x(j)|^p \right)^{\frac{1}{p}}$$

e

$$\|\hat{x}\|_\infty = \max\{|x(j)| : j \in [n]\}$$

**Fatto 1.1** : Per ogni vettore  $\hat{x} = \langle x(1), \dots, x(n) \rangle \in \mathbb{R}^n$  vale che :

$$\|\hat{x}\|_\infty \leq \|\hat{x}\|_2 \leq \|\hat{x}\|_1 \leq \sqrt{n} \cdot \|\hat{x}\|_2 \leq n \cdot \|\hat{x}\|_\infty$$

## Il protocollo di Averaging

Consideriamo il seguente protocollo, chiamato AVG-PROTOCOL :

- **Input** : Ogni nodo  $i$  ha un valore iniziale, chiamato stato,  $x(i) \in \mathbb{R}^+$ 
  - Sia  $\hat{x} = \langle x(1), \dots, x(n) \rangle^T$
- **Ad ogni round  $t \geq 1$  ogni nodo  $i$  fa:**
  - **Pull** : il valore corrente  $x(j)$  da ogni vicino  $j \in N(i)$
  - **Update** : del proprio valore come  $x'(i) = \frac{1}{d_i} \sum_{j \in N(i)} x(j)$

## Analisi del protocollo per Grafi Connessi $d$ -regolari

Si può verificare immediatamente che vale il seguente Claim

**Claim 2.1** : La message complexity per round del protocollo è  $\Theta(m)$

Ora, le domande principali che ci dobbiamo porre sono :

1. I valori  $x(i)$  convergono a un qualche valore  $M$ ?
2. Se sì, cos'è  $M$ ? È la media globale di tutti i valori iniziali?
3. Quanto tempo ci vuole affichè il processo sia "vicino" a una configurazione stabile?

**oss** : la convergenza non viene riconosciuta dai nodi, mentre la terminazione viene riconosciuta da ogni nodo

Per il momento, non consideriamo la terminazione del processo, ma ci concentriamo solo sulla convergenza della sequenza temporale  $\hat{x}^{(0)}, \hat{x}^{(1)}, \dots, \hat{x}^{(t)}, \dots$  dove  $\hat{x}^{(t)} = \langle x^{(t)}(1), \dots, x^{(t)}(n) \rangle^T \in (R^+)^n$  è il vettore colonna rappresentante la configurazione del sistema al tempo  $t$

Ora, sia  $A \in Mat_{(n \times n)}$  la matrice di adiacenza di  $G$  e sia  $D^{-1} \in Mat_{(n \times n)}$  la matrice diagonale, con

$$\begin{aligned} A(i, j) &= 1 \iff (i, j) \in E \\ D^{-1}(i, i) &= \frac{1}{d_i} \quad \forall i \in V \end{aligned}$$

Allora vale il seguente Claim

**Claim 2.2 :** Per ogni round  $t \geq 1$ , la configurazione del sistema al round  $t$  soddisfa la seguente equazione :

$$\hat{x}^{(0)} = \hat{x}, \quad \hat{x}^{(t+1)} = (D^{-1}A)\hat{x}^{(t)} = D^{-1} \cdot (A \cdot \hat{x}^{(t)}) \quad (1)$$

Inoltre, sia  $P = D^{-1}A$  la matrice di transizione del grafo  $G$  (es. **Markov Chain**), allora :

$$\hat{x}^{(0)} = \hat{x}, \quad \hat{x}^{(t+1)} = P\hat{x}^{(t)} = P^t\hat{x} \quad (2)$$

**dim :** Per ogni  $v \in V$ , possiamo scrivere l' $i$ -esima componente dell'equazione 1 come :

$$x^{(t+1)}(i) = \frac{1}{d_i} \cdot \sum_{j=1}^n A(i, j) \cdot x^{(t)}(j) = \frac{\sum_{j \in N(i)} x^{(t)}(j)}{d_i} \quad (3)$$

**Claim 2.3 :** Assumiamo  $G$  connesso e non-bipartito. Allora, l'insieme di **autovalori** di  $P$  sono tutti reali e, scrivendoli in ordine decrescente, vale che :

$$\langle \lambda_1 = 1, \lambda_2 < 1, \dots, \lambda_n \rangle \quad |\lambda_i| \leq 1, \forall i = 1, \dots, n$$

L'algebra lineare ci permette di scrivere il vettore iniziale  $\hat{x}$  come combinazione lineare di basi ortonormali  $\langle \hat{u}_1, \hat{u}_2, \dots, \hat{u}_n \rangle$  di autovalori di  $P$ , come segue :

$$\hat{x} = \alpha_1 \hat{u}_1 + \alpha_2 \hat{u}_2 + \dots + \alpha_n \hat{u}_n$$

Con,  $G$  è  $d$ -regolare,  $P$  simmetrica e i vettori  $\hat{u}_i$  sono mutualmente ortogonali e hanno norma 2 pari a  $\|\hat{u}_i\|_2 = 1$ . Allora, applichiamo l'equazione (2), ottenendo il seguente lemma

**Lemma 2.4 :**

$$\begin{aligned} \hat{x}^{(t+1)} &= P^t \hat{x} = \alpha_1 \lambda_1 \hat{u}_1 + \alpha_2 \lambda_2^t \hat{u}_2 + \dots + \alpha_n \lambda_n^t \hat{u}_n \quad (4) \\ (\text{dato che } \lambda_1 = 1) &= \alpha_1 \hat{u}_1 + \alpha_2 \lambda_2^t \hat{u}_2 + \dots + \alpha_n \lambda_n^t \hat{u}_n \end{aligned}$$

dove  $\alpha_i = \langle \hat{x}, \hat{u}_i \rangle$  sono le proiezioni dei vettori  $\hat{x}$  originali lungo i vettori della base ortonormale. Vale anche che  $|\alpha_i| = |\langle \hat{x}, \hat{u}_i \rangle| \leq M = \sum_j x(j)$

Osserviamo che, quando  $t \rightarrow \infty$ , dato che  $|\lambda_i| < 1 \forall i \neq 1$ , tutti i termini dell'equazione (4)

tranne il primo tendono esponenzialmente velocemente a vettori aventi norma 0. Di conseguenza, abbiamo una risposta positiva alla prima domanda.

 **Teorema 2.5 >**

Iniziando da una configurazione iniziale  $\hat{x}$ , il sistema converge a una configurazione

$$\hat{x}^{(t)} \rightarrow \alpha_1 \hat{u}_1 = M \quad (5)$$

# Lezione 13 ADRC - Ancora su Avergin Dynamics - il caso dei grafi regolari, Tempo di convergenza del protocollo

## Convergenza di grafi regolari

Abbiamo visto nella lezione precedente che il sistema converge a una configurazione

$$\hat{x}^{(t)} \rightarrow \alpha_1 \hat{u}_1 = M$$

Le domande che ora ci poniamo sono :

- Chi è  $\alpha_1$ ?
- Chi è  $\hat{u}_1$ ?

Per la definizione di autovettore dell'autovalore  $\lambda_1 = 1$  dobbiamo avere che :

$$\hat{u}_1 = P \cdot \hat{u}_1 = \begin{bmatrix} \frac{1}{d_1} & 0 & \dots & \frac{1}{d_1} & 0 & \dots & \frac{1}{d_1} \\ 0 & \frac{1}{d_2} & \dots & \frac{1}{d_2} & 0 & \dots & 0 \\ 0 & 0 & \dots & \frac{1}{d_i} & \dots & \frac{1}{d_i} & 0 \\ 0 & 0 & \dots & \frac{1}{d_n} & \dots & 0 & \frac{1}{d_n} \end{bmatrix} \cdot \begin{bmatrix} u_1(1) \\ u_1(2) \\ \dots \\ u_1(i) \\ \dots \\ u_1(n) \end{bmatrix}$$

Consideriamo la convergenza norma 2. Così facendo, se abbiamo un generico 1-autovettore  $\hat{v}_1$ , abbiamo bisogno di normalizzarlo per ottenere :

$$\hat{u}_1 = \frac{1}{\|\hat{v}_1\|_2} \cdot \hat{v}_1 \quad (5)$$

In generale quindi, abbiamo che

$$\hat{x}^{(t)} \rightarrow \alpha_1 \hat{u}_1 = \langle x, \hat{u}_1 \rangle \hat{u}_1 = \frac{1}{\|\hat{v}_1\|_2} \cdot \hat{v}_1 \cdot \left( \sum_j x(j) u_1(j) \right) \quad (7)$$

Possiamo quindi derivare l'autovalore  $\hat{v}_1$  mostrando la seguente affermazione

**Lemma 2.6 :**

Il vettore  $\hat{y} = \left\langle \frac{d_1}{2m}, \dots, \frac{d_n}{2m} \right\rangle$  è un 1-autovettore destro della matrice  $P^T = (D^{-1}A)^T = AD^{-1}$ , ovvero

$$P^T \hat{y} = \hat{y}$$

**dim**

Dato che  $G$  è  $d$ -regolare, sappiamo che  $\forall v \in V \rightarrow |N(v)| = d$

Calcoliamo quindi

$$v^{(1)}(1) = Pv^{(1)} = \frac{1}{d} \sum_{j \in N(i)} \frac{d}{2m} = \frac{1}{d} d \left( \frac{d}{2m} \right) = \frac{d^2}{d2m} = \frac{d}{2m}$$

Questa situazione vale per ogni entrata del vettore  $\hat{v}$ , e di conseguenza

$$\hat{v} = \left\langle \frac{d_1}{2m}, \dots, \frac{d_n}{2m} \right\rangle \quad \blacksquare$$

### Claim 2.7

Un autovettore per  $\lambda_1 = 1$  è il seguente

$$\hat{v}^T = \left\langle \frac{d}{2m}, \dots, \frac{d}{2m} \right\rangle$$

Dall'equazione (7) otteniamo quindi :

$$x^{(t)} \rightarrow \alpha_1 u_1(i) = \frac{1}{\frac{nd^2}{4m^2}} \left( \sum_i x(i) \frac{d}{2m} \right) \frac{d}{2m} = \frac{d^2/4m^2}{nd^2/4m^2} \sum_j x(j) = \frac{\sum_j x(j)}{n} = \text{MEDIA} \quad (9)$$

Dove :

- $\alpha_1 = \langle x, \hat{u}_1 \rangle$
- $\hat{u}_1(i) = \frac{1}{\frac{nd^2}{4m^2}} \frac{d}{2m}$

Quindi vale il seguente teorema

#### ¶ Teorema 2.8 >

Se il grafo  $G$  è connesso, non-bipartito e  $d$ -regolare, allora il protocollo **AVG-PROTOCOL** converge (in norma 2) a una configurazione stabile in cui ogni nodo ottiene, come suo valore, la media  $\frac{\sum_j x(j)}{n}$  del vettore iniziale  $\hat{x}$

## Tempo di convergenza del protocollo

Prima di entrare nel dettaglio dell'analisi, dobbiamo chiarire quale nozione di convergenza per il processo di averaging e quale nozione di output locale dobbiamo considerare.

Come si può osservare, nel codice di **AVG-PROTOCOL** non c'è nessuna regola di stop, nessun valore di output ritornato, e nessun criterio di stop.

Ricordiamo che stiamo assumendo che i nodi del sistema distribuito possono eseguire computazioni algebriche con precisione infinita.

Questo ovviamente non è realistico

Per ovviare a questo problema, aggiungiamo un nuovo paramentro di confidenza in input e un criterio di stop che ci permette di avere un piccolo errore nei valori calcolati-

Il nuovo protocollo è quindi :

### AVG-PROTOCOL1

- **Input** : Ogni nodo  $i$  ha un valore iniziale, chiamato stato,  $x(i) \in \mathbb{R}^+$  (Sia  $\hat{x} = \langle x(1), \dots, x(n) \rangle^T$ ) e un parametro di confidenza  $\varepsilon > 0$
- **Ad ogni round**  $t \geq 1$  ogni nodo  $i$  fa:
  - **Pull** : il valore corrente  $x(j)$  da ogni vicino  $j \in N(i)$
  - **Update** : del proprio valore come  $x'(i) = \frac{1}{d_i} \sum_{j \in N(i)} x(j)$
  - **Se**  $x'(i) - x(i) \leq \varepsilon$  allora **return**  $x'(i)$  e si **ferma**

Iniziamo ad analizzare il protocollo in un qualunque nodo fissato del sistema come segue.

Osserviamo che, dato che stiamo guardando al valore *locale* mantenuto da ogni nodo, dovremmo considerare la norma  $\|\cdot\|_\infty$  che limita il valore **massimale** tra tutti i nodi

Per ogni istante di tempo  $t \geq 1$  vale che :

$$\begin{aligned} \hat{x}^{(t+1)} - \hat{x}^{(t)} &= P^{t+1}\hat{x} - P^t\hat{x} \\ &= \alpha_1\hat{u}_1 - \alpha_1\hat{u}_1 + (1 + \lambda_2)\lambda_2^t\alpha_2\hat{u}_2 + \dots + (1 + \lambda_n)\lambda_n^t\alpha_n\hat{u}_n \quad (10) \\ &= (1 + \lambda_2)\lambda_2^t\alpha_2\hat{u}_2 + \dots + (1 + \lambda_n)\lambda_n^t\alpha_n\hat{u}_n \end{aligned}$$

Definiamo quindi

$$\lambda = \max\{|\lambda_j| : j \geq 2\}, \lambda_{\min} = \{|\lambda_j| : j \geq 2\}$$

e ricordiamo che dato che  $G$  è connesso e non-bipartito allora vale che

$$0 < \lambda_{\min} \leq \lambda < 1$$

Quindi, l'eq (10) implica che

$$\begin{aligned} \|\hat{x}^{(t+1)} - \hat{x}^{(t)}\|_\infty &= \|P^{t+1}\hat{x} - P^t\hat{x}\|_\infty \\ &\leq n(1 - \lambda_{\min})\lambda^t\alpha_n\|\hat{u}_n\|_\infty = n(1 - \lambda_{\min})\lambda^t M \quad (11) \end{aligned}$$

con  $M = \sum_j x(j)$  e osserviamo che, per ogni  $i \in [n]$ ,  $\alpha_i \leq M$

Quindi ora possiamo vedere quanto grande deve essere  $t$  in modo da ottenere l'ultimo "incremento" sufficientemente piccolo del valore di ogni nodo, ovvero :

$$\|\hat{x}^{(t+1)} - \hat{x}^{(t)}\|_\infty \leq \varepsilon \quad (12)$$

Dall'eq. (11) sappiamo che il processo indotto dal protocollo **AVG-PROTOCOL1** fermerà ogni nodo entro  $T$  rounds, tale che  $T$  soddisfa :

$$n(1 - \lambda_{\min})\lambda^T M \leq \varepsilon$$

e quindi

$$\lambda^T \leq \frac{\varepsilon}{nM(1 - \lambda_{\min})}$$

Dato che  $|\lambda| < 1$ , possiamo riscrivere l'equazione come :

$$T \geq \log \left( \frac{nM(1 - \lambda_{\min})}{\varepsilon} \right) / \log \left( \frac{1}{\lambda} \right)$$

e dato che  $|\lambda_{\min}| < 1$ , abbiamo che per ogni

$$T \geq \log \left( \frac{nM}{\varepsilon} \right) / \log \left( \frac{1}{\lambda} \right)$$

vale l'eq. (12)

Abbiamo quindi dimostrato il seguente lemma

### Lemma 2.9 :

Il processo indotto dal protocollo **AVG-PROTOCOL1** termina dopo

$$T = \log \left( \frac{nM}{\varepsilon} \right) / \log \left( \frac{1}{\lambda} \right)$$

e quindi, il valore  $x'(i)$  al round  $T$  di ogni nodo  $i$  sarà  $\varepsilon$ -vicina al valore  $x(i)$  del round precedente.

### ¶ Teorema 2.10 >

Con input un grafo connesso, non-bipartito e  $d$ -regolare e un parametro di confidenza  $\varepsilon > 0$ , il protocollo **AVG-PROTOCOL1** termina dopo  $T = \log \left( \frac{nM}{\varepsilon} \right) / \log \left( \frac{1}{\lambda} \right)$  round.

Inoltre, ogni nodo  $i \in [n]$  ritorna un valore  $x^{(T)}$  tale che

$$\left| x^{(T)}(i) - \frac{\sum_j x(j)}{n} \right| \leq \varepsilon$$

### dim

Dall'equazione (11) sappiamo che

$$\begin{aligned} \|\hat{x}^{(t+1)} - \hat{x}^{(t)}\|_\infty &= \|P^{t+1}\hat{x} - P^t\hat{x}\|_\infty \\ &\leq n(1 - \lambda_{\min})\lambda^t \alpha_n \|\hat{u}_n\|_\infty \quad (13) \\ &\leq n(1 - \lambda_{\min})\lambda^t M \\ &\leq n\lambda^t M \end{aligned}$$

D'altra parte

$$\begin{aligned} \|\hat{x}^{(t)} - \alpha_1 \hat{u}_1\|_\infty &= \|P^t \hat{x} - \alpha_1 \hat{u}_1\|_\infty \\ &= \lambda_2^t \alpha_2 \hat{u}_2 + \cdots + \lambda_n^t \alpha_n \hat{u}_n \quad (14) \\ &\leq n\lambda^t M \|\hat{u}_n\|_\infty = n\lambda^t M \end{aligned}$$

Grazie al Lemma 2.9, e comparando le equazioni (13) e (14) possiamo affermare che quando il nodo  $i$  ferma la sua computazione, dopo  $T$  round, ottiene un valore tale che

$$|x^{(T)}(i) - \alpha_1 \hat{u}_1| = |x^{(t)}(i) - M/n| \leq n\lambda^t M \leq \varepsilon \quad \blacksquare$$

# Lezione 14 ADRC - Introduzione al modello Radio Network, Primo Protocollo con Round Robin

## Modello Radio Networks

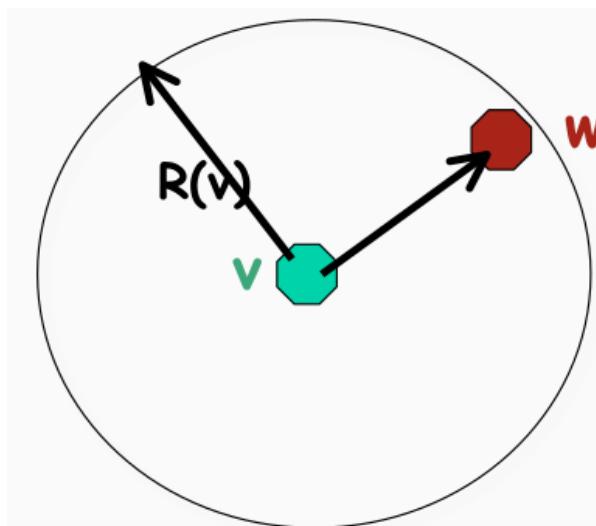
Passiamo dal modello LOCAL al modello RADIO

### Ξ Definizione >

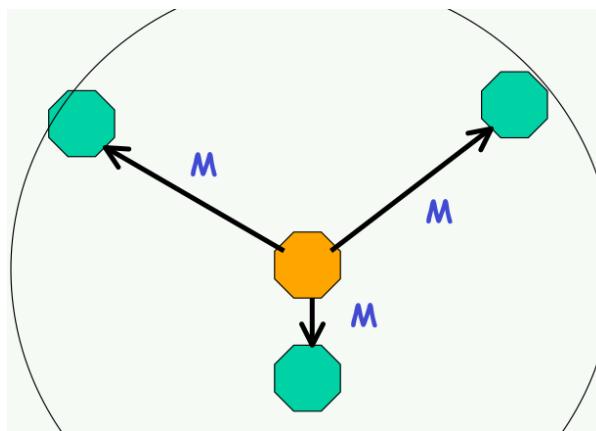
Una **Radio Network** è un insieme di **stazioni** (nodi) posizionati su uno spazio Euclideo

Ad ogni nodo  $v$  viene assegnato un range di trasmissione  $R(v) > 0$

Un nodo  $w$  può ricevere un *messaggio*  $M$  da  $v \iff d(v, w) \leq R(v)$



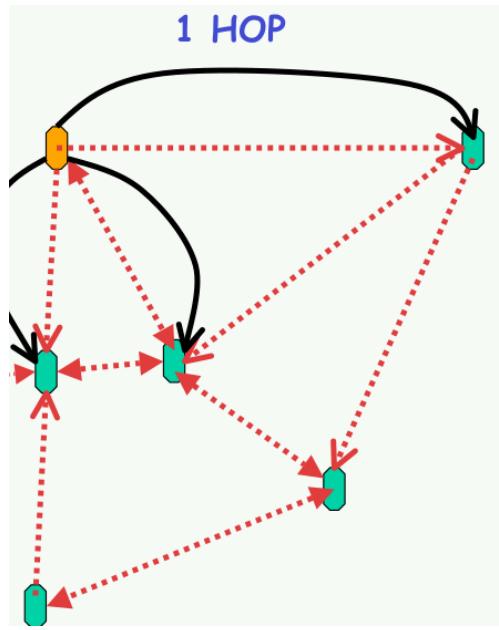
Quando un nodo  $v$  invia un messaggio  $M$ , il messaggio viene inviato su **tutti** gli archi uscenti da  $v$  (**Trasmissione Broadcast**) in un singolo **Time Slot**



Le Radio Networks sono **Sistemi Sincroni**, infatti tutti i nodi condividono lo stesso *clock globale*.

Di conseguenza, i nodi lavorano con Slot di Tempo, e la trasmissione dei messaggi viene completata in un singolo **time slot**

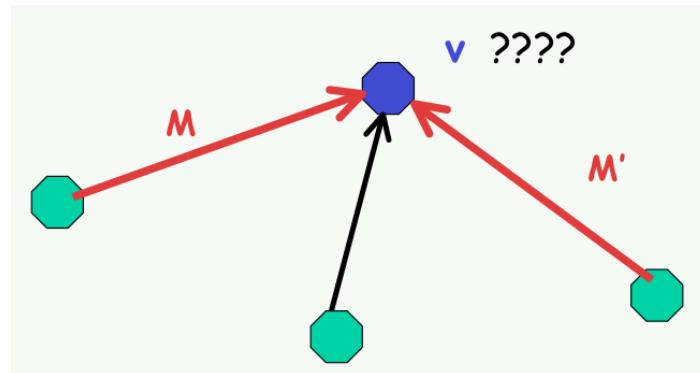
L'assegnamento dei Range determina univocamente un **Grafo Diretto di Comunicazione**  $G(V, E)$



Tutti i vicini entranti di  $s$  (nodo arancione) ricevono il messaggio in 1 salto, a meno che...

## Collisione dei messaggi (interferenza)

Se, durante un time slot, **due o più vicini entranti** inviano un messaggio a  $v$  allora  $v$  **non riceve nulla**



Quindi :

Un nodo  $v$  riceve un messaggio durante lo slot  $T \iff$  c'è **esattamente** un vicino-entrante di  $v$  che invia il messaggio  $M$  durante lo slot  $T$

## Broadcast su Radio Network

Uno dei task più comuni nel mondo Radio è quello di fare broadcast su tutta la rete.

Vogliamo quindi progettare un protocollo che porti a termine correttamente il broadcast, senza causare collisioni di messaggi.

Il problema lo modelliamo in questo modo :

- Configurazione iniziale ( $t = t_0$ ): un solo nodo  $s \in V$  si trova nello stato INFORMED, mentre tutti gli altri nodi  $w \in V - \{s\}$  sono nello stato NON INFORMED
- Configurazione finale : Tutti i nodi  $v \in V$  si trovano nello stato INFORMED

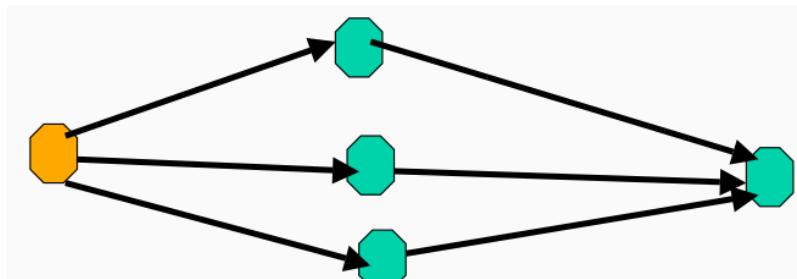
Ricordiamo che per il problema del Broadcast abbiamo :

- **Correttezza** : Un protocollo **completa** il Broadcast dalla sorgente  $s$  a tutto il grafo  $G$  se c'è un time slot tale che ogni nodo si trova nello stato INFORMED
- **Terminazione** : Un protocollo **termina** se c'è un time slot  $t$  tale che **ogni** nodo **interrompe** ogni sua azione ENTRÒ il time slot  $t$

## Protocollo FLOOD

Vediamo inizialmente il protocollo FLOOD studiato qualche tempo fa per la risoluzione del Broadcast sul modello LOCAL

È facile vedere subito che il FLOOD non funziona, infatti basta prendere un grafo fatto in questo modo



Dove il nodo arancione è la sorgente  $s$

All'istante  $t = t_0$  la sorgente  $s$  invia il messaggio ai suoi 3 vicini, siano essi per semplicità  $u, v, w$ . Subito dopo, i rispettivi nodi inviano il messaggio al loro unico vicino, sia esso  $x$ . Risulta però che nello stesso istante di tempo  $t = t_1$  il nodo  $x$  riceve 3 messaggi, provenienti rispettivamente da  $u, v, w$ , e di conseguenza avviene la collisione, causando così la perdita del messaggio per  $x$  ( $x$  non riceve nulla).

Con questo semplice controesempio abbiamo visto che il protocollo FLOOD non soddisfa le condizioni necessarie per evitare collisioni, e di conseguenza dobbiamo pensare a un protocollo diverso.

## Protocollo Round-Robin=

Il protocollo che vedremo ora sfrutta l'idea generale del famoso algoritmo di scheduling **Round-Robin**

Prima di vedere il protocollo è necessario definire le assunzioni che stiamo ponendo :

- I nodi conoscono una buona approssimazione di  $|V| = n$
- I nodi sono indicizzati partendo da  $0, 1, 2, \dots$

A questo punto il protocollo lavora nel modo seguente, a **fasi**

- Una **fase** di RR è composta da  $n$  time slot
- A tempo  $T = 0, 1, 2, \dots$ 
  - Se il nodo  $i = T$ , se **INFORMATO**, trasmette il messaggio  $M$  ai suoi vicini
  - Tutti gli altri nodi non inviano nulla

Osserviamo che per come è fatto questo protocollo, esso non terminerà mai (problema che verrà affrontato poco più avanti)

**Q.** Cosa possiamo dire dopo una singola fase di RR?

**A** Dopo la prima fase di RR ( $n$  time slot) **TUTTI** i vicini-uscenti di  $s$  (sorgente) saranno informati

Eseguiamo il protocollo per  $L$  volte consecutive

Vale quindi il seguente teorema :

### ✖ Teorema >

Dopo la fase  $k$ , tutti i nodi a distanza di salto (hop-distance)  $k$  dalla sorgente  $s$  saranno nello stato INFORMED

La dimostrazione è per induzione sulla fase  $k$

**dim**

Caso base : nella prima fase, il primo nodo a trasmettere sarà la sorgente  $s$  nel rime slot  $T = s$ , inoltre in quel time slot  $s$  è l'unico nodo a trasmettere (in quanto è l'unico nodo a possedere il messaggio ed essere nello stato INFORMED).

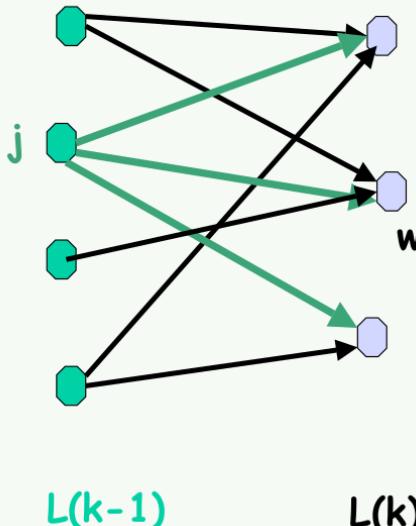
Allora, per costruzione del grafo di comunicazione, tutti i nodi a distanza 1 da  $s$  entro la fine della prima fase saranno nello stato INFORMED

Caso *induttivo*: ipotizziamo che la tesi sia vera per i nodi dell'insieme  $L(k - 1)$ , ovvero tutti i nodi a distanza  $(k - 1)$  da  $s$ .

Sia  $w \in L(k)$ , per definizione stessa di  $L(k)$  deve esistere almeno un nodo, sia esso  $j$ , che appartiene a  $L(k - 1)$ , quindi  $j \in L(k - 1)$ , tale per cui esiste l'arco diretto  $(j, w)$

Per ipotesi induttiva, al time slot  $j \pmod{n}$  (ovvero la fine della fase  $k - 1$ ) il nodo  $j$  trasmetterà il messaggio al nodo  $w$ , e per definizione del protocollo RR il nodo  $j$  sarà l'unico a trasmettere in quell'istante di tempo, quindi il nodo  $w$  riceverà correttamente il messaggio senza subire interferenze, entro la fine della fase  $k$

### Informed Nodes



Questa assunzione vale  $\forall j \in L(k-1)$ , quindi tutti i nodi  $w \in L(k)$  verranno informati entro la fine della fase  $k$  ■

### ⇒ Corollario (Tempo di Completamento) >

Sia  $D$  l'**eccentricità** (sconosciuta) della sorgente  $s$ . Allora,  $D$  fasi di RR sono *sufficienti* per INFORMARE tutti i nodi

Cosa possiamo dire della terminazione?

- Purtroppo dipende dalla conoscenza dei nodi

Se loro conoscono  $n$  allora posso decidere quando fermarsi.

L'eccentricità della sorgente è al più  $n - 1$  quindi, dato che ogni nodo ha un clock globale, i nodi possono decidere di **fermare** la loro esecuzione dopo la  $(n - 1)$ -esima fase di RR

Infatti vale :

### ⇒ Tempi per protocollo RR >

Il protocollo **completa** il task in tempo  $\Theta(Dn)$

Il protocollo **termina** (e si parla di terminazione globale) in tempo  $O(n^2)$

# Lezione 15 ADRC - Ancora su Radio Networks, protocollo "selettivo" per Broadcast

## Continuo protocollo Round Robin

### Non-Conoscenza di $|V|$

Fin'ora abbiamo assunto che i nodi conoscessero una buona approssimazione di  $|V| = n$ , ma cosa succede se i nodi non conoscono  $n$ ?

Devono in qualche modo "indovinarlo", sfruttando l'idea della **Ricerca Binaria**, e poi simulare il protocollo *RR* usando il valore "indovinato".

Adremo quindi a incrementare il valore di  $n$  in maniera *esponenziale*

Vediamo allora come funziona questo nuovo protocollo, chiamato **GEN-RR(L)**

#### **GEN-RR(L):**

For  $L = 2, 3, \dots$  DO

- For  $T = 1, 2, \dots, 2^L$ :
- Ogni nodo  $i$  fa:
  - if  $i \equiv T \pmod{2^L} \wedge \text{status}=\text{INFORMED}$
  - Trasmette e imposta il suo status a DONE (Do Nothing)

Analizziamo il seguente protocollo :

Sia  $l = \min\{L : 2^L \geq n\}$ , allora

$$TIME(GEN - RR(l)) = 2^l \cdot D$$

Di conseguenza, avremo che

$$TIME(GEN - RR(n)) = \sum_{l=1}^{\lceil \log(n) \rceil} 2^l D = D \sum_{l=1}^{\lceil \log(n) \rceil} \leq D \sum_{l=1}^{\lceil \log(n) \rceil} n = (Dn) \log(n)$$

Anche qui,  $D$  potrà essere al più  $n - 1$ , quindi la complessità temporale del protocollo **GEN-RR(n)** sarà  $O(((n - 1)n) \log(n)) = O(n^2 \log(n))$

## Protocollo "selettivo"

Vediamo ora un'altro protocollo per il problema del Broadcast, che sfrutta un'osservazione fondamentale.

**oss** : il protocollo RR non *sfrutta per niente* il parallelismo

Vediamo quindi il metodo "selettivo"

### DEFINIZIONE (Famiglia $(n,k)$ -selettiva) >

Dato  $[n] \in \{1, 2, \dots, n-1\}$  e  $k \leq n$ , una famiglia di sottoinsiemi

$$\mathcal{H} = \{H_1, H_2, \dots, H_t\}$$

è detta  $(n, k)$ -selettiva se

$$\forall S \subseteq [n] : |S| \leq k \implies \exists H \in \mathcal{H} : |S \cap H| = 1$$

Una famiglia  $(n, k)$ -selettiva banale è la famiglia composta da tutti **singleton**, ovvero sottoinsieme composti da un singolo elemento, come segue :

$$\mathcal{H} = \{\{1\}, \{2\}, \dots, \{n\}\}$$

Detto questo, come può una famiglia selettiva essere usata per il problema del Broadcast?

Mettiamoci nella restrizione che tutti i nodi conoscono  $n$  e  $d$ , allora :

**set-up** : tutti i nodi conoscono la stessa famiglia  $(n, d)$ -selettiva

$\mathcal{H} = \{H_1, H_2, \dots, H_i, \dots, H_t\}$  dove  $d = \text{massimo grado}(G)$

Il protocollo sarà il seguente :

### Protocollo SELECT1

- Lavora in fasi consecutive  $J = 1, 2, \dots$  (come RR)
- Al time slot  $i$  di ogni fase, ogni **nodo informato**  $\in H_i$  trasmette il messaggio

Analizziamo il protocollo, vale il seguente lemma

**Lemma 1** : Dopo la fase  $j$ , tutti i nodi a distanza al più  $j$  saranno informati

**dim** : per induzione su  $j$

- *Caso base* :  $j = 0$  solo la sorgente sarà informata (come RR) e di conseguenza sarà l'unica a trasmettere
- *Caso induttivo* : Consideriamo un nodo  $y$  a distanza  $j$ . Consideriamo il sottoinsieme

$$N(y) = \{z \in V : z \text{ è vicino di } y \wedge z \text{ si trova a distanza } j-1\}$$

Dato che  $N(y) \subseteq [n]$  e  $|N(y)| \leq n$ , se applichiamo la  $(n, d)$ -selettività otteniamo la tesi

- Infatti,  $\forall v \in N(y) \implies |N(v) \cap N(y)| \leq 1$

Ma è corretto questo protocollo? La risposta è **NO**

Infatti non stiamo considerando l'impatto dei nodi  $z$  **informati** nel livello  $j$  durante la fase  $j$   
Valgono infatti due casistiche :

1. se poniamo  $z \in N(y)$ ,  $z$  potrebbe essere selezionato ma non ancora informato
2. se non poniamo  $z \in N(y)$ ,  $z$  potrebbe essere informato e creare **collisioni**

Come possiamo sistemare la situazione? Usando una *semplifica* modifica

"Solo i nodi che sono stati informati DURANTE la fase  $j - 1$  saranno nello stato ACTIVE durante la fase  $j$ "

Adesso, il lemma1 è vero, quindi dopo  $D$  fasi tutti i livelli  $L_0, \dots, L_D$  saranno informati

Quindi, il **tempo di completamento** del protocollo SELECT1 è pari a  $O(D \cdot |\mathcal{H}|)$ , dato che ogni fase impiega tempo  $|\mathcal{H}|$ .

Ci serve quindi una famiglia selettiva **min-size**. Vale quindi il seguente teorema, di cui non faremo la dimostrazione :

### Teorema (ClementiMontiSilvestri) >

Per un  $n$  e  $k \leq n$  sufficientemente grandi :

$$\exists \mathcal{H} = \{H_1, H_2, \dots, H_t\} : \mathcal{H} \text{ è } (n, k)\text{-selettiva} \wedge |\mathcal{H}| = \Theta(k \log(n)) \ll n \log(n)$$

ed è ottimale

Quindi, se mettiamo questa famiglia selettiva all'interno del protocollo otteniamo che

$$\text{COMPL-TIME(SELECT1)} = O(D \cdot d \log(n))$$

Quindi se  $D$  e  $d$  sono entrambi **piccoli**, abbiamo un tempo di completamento **molto migliore** del RR

# Lezione 16 ADRC - LowerBound al protocollo seletivo, protocollo Random

## Lower Bound al protocollo selettivo

Il protocollo selettivo può essere **migliorato** nei grafi generali? La risposta è no, infatti vale il seguente teorema

### Lower Bound >

Nei **grafo generali diretti**, l'uso di una famiglia selettiva è in qualche modo **necessaria**. Infatti vale che

$$\forall Dd \leq n \implies \Omega\left(Dd \log\left(\frac{n}{d}\right)\right)$$

Non faremo la dimostrazione di questo teorema, però è un risultato forte perchè ci afferma che meglio di  $Dd \log \frac{n}{d}$  non possiamo fare.

## Random vs Deterministic : Gap Esponenziale

Prendiamo un grafo di esempio con  $d \simeq n, D = c$  costante.

Abbiamo visto quindi che il lower bound per i protocolli deterministicici è  $\Omega(n \log(n))$

Cosa possiamo dire dei protocolli **randomizzati**?

Facciamo un breve esempio di protocollo randomizzato

## Protocollo probabilistico semplice

Ad ogni istante di tempo, ogni nodo informato trasmette con probabilità  $\frac{1}{2}$  (lancio di moneta, indipendente)

- Abbiamo quindi che,  $\forall v \in V, \forall t \rightarrow X_v^t = \begin{cases} 1 & \text{Esce Testa, il nodo trasmette} \\ 0 & \text{altrimenti} \end{cases}$

Analizziamo questo protocollo su un grafo molto semplice, composto da 3 livelli  $L_0, L_1, L_2$

- $s \in L_0$
- $|L_1| = \frac{n}{2} - 1$  nodi
- $|L_2| = \frac{n}{2}$  nodi,  $y \in L_2$

Ipotizziamo che tutti i nodi in  $L_1$  siano attivi e pronti a trasmettere

Vediamo che  $T \leq T_1 + T_2$

tempo per  $L_1$       tempo per  $L_2$

È facile vedere che  $\mathbb{E}[T_1] = \frac{1}{p} = 2$  perché  $p = \frac{1}{2}$

Per  $\mathbb{E}[T_2]$  usiamo l'evento  $\mathcal{E}(y) = "y$  viene informato". Abbiamo che

$$\Pr[\mathcal{E}(y) | L_1] = \Pr_{\substack{\text{nodo trasmette esce Croce per } \frac{n}{2}-1 \text{ nodi} \\ \text{tutti i nodi in } L_1 \text{ informati}}} \left( \frac{\frac{n}{2}-1}{1} \right) \left( \frac{1}{2} \right)^{\frac{n}{2}-1} \\ = \left( \frac{n}{2} - 1 \right) \left( \frac{1}{2} \right)^{\frac{n}{2}-1}$$

Quindi

$$\Pr[\mathcal{E}(y)^t | L_1] \leq O\left(\frac{n}{2^{\frac{n}{3}}}\right) \sim \text{Geo}\left(p = \frac{n}{2^{\frac{n}{3}}}\right)$$

E di conseguenza

$$\mathbb{E}[T_2] = \frac{1}{p} = \Theta\left(\frac{2^{\frac{n}{3}}}{n}\right)$$

che è **esponenziale**, e di conseguenza non va bene.

Vediamo ora un'approccio sicuramente migliore

## Protocollo probabilistico BGI

Mettiamoci nel caso di grafi  $d$ -regular layered (ovvero grafi che possono essere organizzati a livelli  $L_0, L_1, \dots, L_D$  dove tutti gli archi vanno da qualsiasi nodo di un livello  $j$  ad un qualsiasi nodo del livello  $j+1$ , e il grado entrante di ogni nodo è esattamente  $d$ )

Il protocollo **RND BGI** è il seguente :

For  $K = 1, 2, \dots$  (stages)

- For  $j = 1, 2, \dots, c \log(n)$ 
  - If nodo  $x$  è **stato informato** nello stage  $K-1$  allora trasmette con probabilità  $\frac{1}{d}$

Analizziamo il protocollo

Vale il seguente teorema :

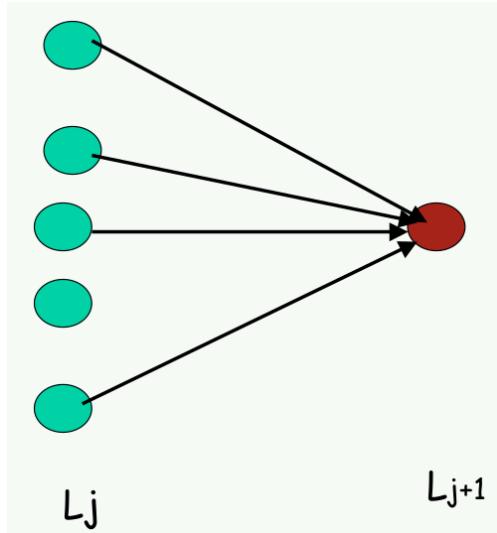
 Teorema >

Il protocollo BGI completa il Broadcast entro  $O(D)$  stages, e dato che ogni stage richiede tempo  $\text{O}(\log(n))$ , il tempo di completamento è  $O(D \log(n))$  w. h. p

**dim** Per induzione sui livelli  $L = 1, \dots, D$

Caso base : Come sempre, la sorgente è l'unico nodo informato. Possiamo assumere che essa trasmetta con probabilità 1

Caso *induttivo* : Per ipotesi induttiva, assumiamo che entro lo stage  $j > 1$  tutti i nodi in  $L_j$  siano informati. Mostriamo che è vero anche per  $j + 1$



Sia  $y \in L_{j+1}$ , vediamo la probabilità che  $y$  venga informato nel time slot della fase  $j + 1$ .

Per definizione di  $d$ -regular layered,  $y$  ha esattamente  $d$  vicini nel livello  $L_j$ , i quali sono tutti informati per ipotesi induttiva

Il nodo  $y$  viene informato se e solo se, tra tutti i suoi vicini  $d$  solo un trasmette e gli altri no.

Prendiamo in considerazione l'evento

$$\mathcal{E} = \text{"y viene informato in un timeslot"}$$

Allora, la probabilità che  $y$  venga informato è

$$\begin{aligned} Pr[\mathcal{E}] &= \binom{d}{1} \frac{1}{d} \left(1 - \frac{1}{d}\right)^{d-1} \\ &= \left(1 - \frac{1}{d}\right)^{d-1} \geq \frac{1}{8} \quad \forall d \geq 2 \end{aligned}$$

Notiamo che  $\frac{1}{8}$  è una costante assoluta che non dipende da  $n, D, d$

Sia ora  $\mathcal{E}' = \text{"y non viene informato dopo } c \log(n)\text{"}$

La probabilità questo evento accada è (**ricordando che i timeslot sono tutti indipendenti**)

$$\Pr[\mathcal{E}'] < \left(1 - \frac{1}{8}\right)^{c \log(n)} < e^{-\frac{c \log(n)}{8}} < \frac{1}{n^{\frac{c}{8}}}$$

la seconda disegualanza è vera perché  $(1 - x) < e^{-x}$

Però questa è la probabilità che **un nodo** non venga informato. Per vedere la probabilità che **tutti** i nodi in  $L_{j+1}$  vengano informati, dobbiamo usare lo **UnionBound**

Quindi vale che

$$\Pr[\exists x \in L_{j+1} : x \text{ non viene informato entro lo stage } j+1] \leq n \left( \frac{1}{n^{\frac{c}{8}}} \right) \leq \frac{1}{n^{\frac{c}{8}-1}}$$

A questo punto, dobbiamo far vedere che per ogni livello  $k = D < n$ , tutti i nodi in  $L_k$  sono informati entro lo stage  $k$  con altra probabilità.

Per fare ciò, si calcola la probabilità dell'evento  $\mathcal{B} = \exists \text{BAD STAGE}$ , cioè l'evento tale per cui esiste uno stage in cui non tutti i nodi del relativo livello vengono informati.

Anche qui applichiamo lo UnionBound su tutti i livelli.

$$\Pr[\mathcal{B}] < \sum_{j=1}^n \Pr[\exists x \in L_j : x \text{ non viene informato entro lo stage } j] < n \left( \frac{1}{n^{\frac{c}{8}-1}} \right) = \frac{1}{n^{\frac{c}{8}-2}}$$

# Lezione 17 ADRC - Fine parte su prot. RND per R.N, Introduzione al modello GOSSIP

## Continuo prot. RND per R.N

### Estendere BGI a grafi generali

Possiamo estendere il protocollo BGI a **grafi generali** per completare il Broadcast in tempo  $O(D \log^2(n))$  w.h.p

Per fare ciò dobbiamo metterci nella restrizione che i nodi conoscano  $n$

Modifichiamo il protocollo BGI in questo modo, aggiungendo un ulteriore ciclo for :

For  $L = 1, \dots, \lceil \log(n) \rceil$

- Nodo trasmette con probabilità  $\frac{1}{2^L}$

Il protocollo modificato sarà quindi :

For  $K = 1, \dots$  (stages)

- For  $L = 1, \dots, \lceil \log(n) \rceil$ 
  - For  $j = 1, 2, \dots, c \log(n)$ 
    - Se nodo  $x$  è stato informato in stage  $k - 1$ ,  $x$  trasmette con probabilità  $\frac{1}{2^L}$

Vediamo che :

- Il ciclo for interno costa  $O(\log(n))$
- Il secondo ciclo for costa anch'esso  $O(\log(n))$
- Il ciclo for più esterno costa  $O(D) = \text{diam}(G)$

In totale abbiamo che

$$\text{Time}(BGI - Mod) = O(D \log^2(n))$$

Se i nodi non conoscono  $n$ , possono "indovinarne" il valore usando la ricerca **binaria**

- Così facendo però i nodi non possono terminare
- Il tempo di completamento diventa  $O(D \log^3(n))$  w.h.p

## Modelli GOSSIP

Abbandoniamo il modello RADIO per introdurre una nuova classe di modelli, ovvero i modelli **GOSSIP**. (nello specifico **GOSSIP PUSH e PULL**)

Per un grafo non diretto  $G = (V, E)$ , con  $|V| = n$  e  $|E| = m$ , denotiamo il vicinato di un nodo  $v \in V$  come  $N(v)$  e  $d(v) = |N(v)|$

### Definizione (Il modello GOSSIP PUSH (PULL)) >

Dato un (non diretto) grafo  $G(V, E)$ , il modello di comunicazione **uniforme PUSH (PULL)** lavora in modo sincrono, con round discreti.

Ad ogni round  $t = 0, 1, \dots$  ogni nodo  $v \in V$  **sceglie** u.a.r uno dei suoi vicini  $u \in_u N(v)$  ed esegue l'operazione di **push (pull)** per prendere (inviare) un qualunque messaggio  $M$  da (a)  $u$

Alla fine di ogni round  $t$ , il nodo  $u$  (nodo  $v$ ) avrà il messaggio  $M$

## Proprietà dei modelli GOSSIP

Ad ogni round, il numero totale di comunicazioni (es. trasmissioni dei messaggi) è  $n$  in entrambi i modelli.

In particolare, queste comunicazioni generano un grafo diretto delle comunicazioni al round  $t$  chiamato  $G_t(V_t, E_t)$ , dove  $E_t$  rappresenta l'insieme di **archi attivi** ad ogni round  $t$ .

Il grafo  $G_t$  è sempre un **grafo sparso**.

Valgono quindi le seguenti proprietà

1. Nel modello **PUSH**, ad ogni round, il numero atteso di operazioni *push ricevute* da ogni nodo è 1, e in totale è  $O(\log(n))$  w.h.p
2. Nel modello **PULL**, ad ogni round, il numero atteso di operazioni *pull ricevute* da ogni nodo è 1, e in totale è  $O(\log(n))$  w.h.p

## Protocollo PULL su Clicque

Analizziamo il seguente PULL Broadcast Protocolo (**BP**) sul grafo completo  $K(V, E)$

Sia  $M$  l'informazione che ogni nodo deve possedere per completare il task del Broadcast. Assumiamo che ogni entità  $x \in V$  abbiamo un registro privato  $c_x$  tale che :

$$c_x = \begin{cases} \text{informed} & x \text{ ha ricevuto } M \\ \text{not-informed} & x \text{ non ha ricevuto } M \end{cases}$$

In modo formale, il problema è descritto dalla tripla  $\langle P_{init}, P_{final}, G_{pull} \rangle$ , dove :

- $P_{init} = \exists! x \in V : c_x = \text{informed} \wedge \forall y \neq x, c_y = \text{not-informed}$
- $P_{final} = \forall x \in V : c_x = \text{informed}$
- $G_{pull} = \text{Restrizioni del modello GOSSIP} \cup K_T$

- $KT$  =Knowledge Topology

Vediamo ora il protocollo effettivo

Tutti i nodi, durante l'esecuzione del protocollo possono trovarsi nei due stati possibili, ovvero {informed,not-informed}

BP su  $K_n, s \in V$ :

- Inizialmente la sorgente  $s$  è l'unica nello stato informed, in quanto è l'unica a possedere il messaggio  $M$ . Tutti gli altri nodi si trovano nello stato not-informed
- Ad ogni round  $t \geq 1$  ogni nodo  $v$  che si trova nello stato not-informed esegue una operazione di **pull** su un'altro nodo  $u \in_u N(v)$  :
  - Se  $u$  è un nodo nello stato informed, allora  $v$  si fa inviare una copia di  $M$  e passa allo stato informed
- Il protocollo termina globalmente quando tutti i nodi si trovano nello stato informed

Vale quindi il seguente teorema

### ✖ Teorema 3.3 >

Il tempo di completamento di BP su input  $(K(V, E), s \in V)$  è  $\Theta(\log(n))$  w.h.p

## Dimostrazione

Fissiamo un qualunque  $t \geq 1$ , e per ogni  $v \in V$  consideriamo la v.a. binaria

$$Y_v = \begin{cases} 1 & \iff v \text{ sarà informato al round } t+1 \\ 0 & \text{altrimenti} \end{cases}$$

Definiamo inoltre

$$I_t = \{u \in V | u \text{ è informato al round } t\}; \quad I_0 = \{s\}$$

Dato che, in ogni round, ogni nodo effettua una operazione di **pull** da un'altro nodo della rete scelto u.a.r, e i nodi informati al round  $t$  sono in totale  $|I_t| = m_t$ , vale che per ogni  $v \in V \setminus I_t$

$$\mathbb{E}[Y_v] = Pr[T_v = 1] = \frac{m_t}{n}$$

A questo punto, la dimostrazione si divide in due parti :

1. **Prima Fase** : Dimostreremo con alta probabilità che il numero di nodi informati al round  $t$  cresce **esponenzialmente** fino a raggiungere  $|I_t| = \frac{n}{2}$
2. **Seconda Fase** : Dimostreremo con alta probabilità che il numero di nodi restanti non informati decresce **esponenzialmente** fino a raggiungere lo zero.

**Prima Fase** :  $m_t \leq \frac{n}{2}$

Fintanto che  $m_t \leq \frac{n}{2}$  abbiamo che

$$\mathbb{E}[m_{t+1} | I_t = m_t] = m_t + \sum_{u \in V \setminus I_t} \frac{m_t}{n} \geq m_t + (n - m_t) \frac{m_t}{n} \geq \frac{3}{2} m_t \quad (2)$$

Questa disegualanza ci dice che, ad ogni round, il numero di nodi informati cresce almeno di un **fattore costante** in media.

In particolare, srotolando l'espressione si ottiene che

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

Se  $\tau = \min\{t \geq 1 : m_t > \frac{n}{2}\}$  è il primo round in cui il numero di nodi informati supera la metà del numero di nodi totali, secondo la relazione di ricorrenza si ottiene che

$$\tau \simeq \log_{\frac{3}{2}} \left( \frac{n}{2} \right) \in \Theta(\log(n))$$

Ovvero in un numero logaritmico di round, almeno la metà dei nodi viene informato, **in media**

Notiamo però che l'analisi che abbiamo appena fatto indica che tutti ciò accade in media. Abbiamo bisogno quindi di prendere i risultati di **concentrazione** ad ogni round usando i bound di **Chernoff**, e poi applicare l'UnionBound su una finestra temporale di lunghezza logartmica.

Dobbiamo quindi dividere la Fase Uno in due sottofasi

**SottoFase 1.1**  $1 \leq m_t \leq \alpha \log(n)$

In questa prima fase, chiamata *Bootstrap*, facciamo riferimento al range  $1 \leq m_t \leq \alpha \log(n)$  per una certa costante  $\alpha > 0$ .

Qui il nostro goal è quello di dimostrare il seguente Claim

### Ξ Claim 3.4 >

Per ogni  $\alpha > 0$ ,  $\exists \gamma = \gamma(\alpha)$  t.c dopo i primi  $\tau_1 = \gamma \log(n)$  round vale che  $m_{\tau_1} \geq \alpha \log(n)$  w.h.p

**dim**

Per ogni  $u \in V \setminus \{s\}$ , consideriamo la v.a

$$Y_u^{(\tau_1)} = 1 \iff u \text{ è informato al round } \tau_1 + 1$$

Essendo che ogni operazion di pull, fatta da ogni nodo al round  $t$ , è **indipendente** rispetto

all'operazione di pull fatta al round  $t + 1$ , vale che :

$$\begin{aligned} Pr\left(Y_u^{(\tau_1)} = 0\right) &= Pr\left(\bigcap_{t=1}^{\tau_1} u \text{ sceglie nodo senza info. al round } t\right) \\ &= \prod_{t=1}^{\tau_1} \left(1 - \frac{m_t}{n}\right) \\ &\leq \left(1 - \frac{1}{n}\right)^t \leq e^{-\frac{\gamma \log(n)}{n}} \end{aligned}$$

dato che almeno la sorgente è informata dall'inizio.

Ora, sia

$$Y^{(\tau_1)} = \sum_{u \in V} Y_u^{(\tau_1)}$$

Vale quindi la seguente disegualanza :

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

Inoltre, dato che le v.a  $Y_u^{(\tau_1)}$  sono **mutualmente indipendenti**, possiamo applicare il **Chernoff bound moltiplicativo** per una costante  $\gamma = \gamma(\alpha) > 0$  suff. grande, in modo da ottenere che, dopo  $\tau_1 = \gamma \log(n)$  round vale che  $m_{\tau_1} \geq \alpha \log(n)$  w.h.p ■

**SottoFase 1.2 :**  $\alpha \log(n) \leq m_t \leq \frac{n}{2}$

La seconda fase, che inizia subito dopo la fase di Bootstrap, si riferisce al periodo in cui  $\alpha \log(n) \leq m_t \leq \frac{n}{2}$

in questa fase dimostreremo il seguente lemma

### ✖ Claim 3.5 >

$\exists \beta$  suff. grande t.c dopo altri  $\tau_2 = \beta \log(n)$  round vale che  $m_t \geq \frac{n}{2}$  w.h.p

**dim**

Dobbiamo procedere round-per-round.

Fissiamo un round  $t \geq \tau_1$ , e sia  $m_t = I_t$  (notiamo che  $I_t$  non è una v.a).

Allora, per ogni  $u \in V \setminus I_t$ , consideriamo la v.a

$$Y_u^{(\tau_1)} = 1 \iff u \text{ è informato al round } \tau_1 + 1$$

Grazie all'equazione (2) sappiamo che

$$\mathbb{E}[m_{t+1}] \geq \frac{3}{2} m_t \geq \alpha \log(n)$$

Inoltre, le v.a  $Y_u^{(\tau_1)}$  sono mutualmente indipendenti, e quindi possiamo applicare il Chernoff Bound nel seguente modo

$$\Pr\left(m_{t+1} \leq (1-\delta)\frac{3}{2}m_t\right) \leq e^{-\frac{\delta^2}{2}\frac{3}{2}m_t} \leq e^{-\frac{\delta^2}{2}\alpha \log(n)} = n^{-\frac{\delta^2}{2}\alpha} = n^{-c}$$

Ponendo  $\delta \geq \frac{1}{3}$  si ottiene che

$$\Pr(m_{t+1} \leq m_t) \leq n^{-c}$$

E di conseguenza si ottiene che

$$\Pr(m_{t+1} > m_t) \geq 1 - n^{-c} \quad (3)$$

A questo punto, per ottenere il claim consideriamo l'evento :

$$\mathcal{E} = \exists \tau_1 \leq t \leq \tau_2 : m_{t+1} < (1.5 - \delta)m_t$$

A questo punto applichiamo lo UnionBound, e otteniamo che la probabilità che questo evento  $\mathcal{E}$  avvenga è :

$$\sum_{t=\tau_1}^{\tau_2} n^{-c} \leq \sum_{t=\tau_1}^{\tau_2} n^{-2} < (\tau_2 - \tau_1)n^{-2} < \frac{1}{n}$$

## Seconda Fase : da $\frac{n}{2}$ a $n$

La seconda fase è del tutto analoga alla prima, con la differenza che si dimostra che la decrescita dei nodi non informati è esponenziale nel tempo.

Si fissi un round  $t \geq \tau_2 = \beta \log(n)$ . Essendo che dalla prima fase si ha che  $m_{\tau_2} \geq \frac{n}{2}$ , si ha che un nodo NOT-INFORMED al tempo  $t$  fa pull del messaggio e diventa INFORMED al tempo  $t + 1$  con probabilità almeno  $\frac{1}{2}$

Sia  $z_t = n - m_t$  il numero di nodi ancora non informati al tempo  $t$ , e sia per ogni nodo  $v \in V \setminus I_t$  non informato  $X_v^{(t)}$  la variabile aleatoria che vale 1 se  $v$  rimane non informato al round  $t + 1$  e 0 altrimenti. Allora, dato che  $z_t \leq \frac{n}{2}$

$$\mathbb{E}\left[X_v^{(t)} | z_t\right] = \Pr\left(X_v^{(t)} = 1\right) = \frac{z_t}{n} \leq \frac{1}{2}$$

Sia  $z_{t+1}$  il numero di nodi non informati al round  $t + 1$ .

Vale che

$$z_{t+1} \sum_{v \in V \setminus I_t} X_v^{(t)}$$

Allora, mediamente, si ha che il numero di nodi ancora non informati al round  $t + 1$  è pari a

$$E[z_{t+1} | z_t] \leq \frac{z_t}{2} \leq \frac{n}{4}$$

Da qui in poi applichiamo il Chernoff Bound per ottenere i risultati in concentrazione.



# Lezione 18 ADRC - Expanders e le loro Proprietà

## Expanders

Proprietà fondamentali nella Network Theory **riguardano** il diametro e la fault-tolerance di un grafo  $G$

È chiaro che vogliamo che  $G$  abbia un **diametro piccolo** e una **buona connettività** anche se qualche arco non funziona.

Per analizzare le due proprietà di cui sopra, dobbiamo introdurre un **concetto fondamentale** in teoria dei grafi :

### Definizione 4.1 >

Dato un grafo  $G(V, E)$   $\Delta$ -regolare, con  $|V| = [n]$ , la **(node)-expansion** di un sottoinsieme  $S \subset [n]$  è definita come :

$$|N(S)| - N(S) = \{w \in V \setminus S : (v, w) \in E \text{ per qualche } v \in S\}$$

Allora, per un  $\alpha > 0$  fissato, diciamo che un grafo  $G$  è detto  $\alpha$ -expander **se ogni sottoinsieme**  $S \subset [n]$ , con  $|S| \leq n/2$ , ha espansione almeno pari a  $\min\{n, \alpha|S|\}$

Il nostro interesse nei grafici  $\Omega(1)$ -expander è ben motivato dal seguente teorema

### Teorema 4.2 >

Consideriamo una famiglia di grafici infinita di dimensione crescente, ovvero

$$\{G_n(V_n, E_n) : |V_n| = [n], n \geq 1\}$$

Se esiste una costante assoluta  $\alpha > 0$  tale che, per un  $n$  suff. grande, il grafo  $G_n$  è un  $\alpha$ -expander allora il suo **diametro** è  $O(\log(n))$ .

Inoltre, sotto le assunzioni di cui sopra, per **scollegare completamente** ogni sottoinsieme  $S$  dal resto del grafo, il numero di **fault links** deve essere almeno lineare nella size di  $S$

**dim**

Consideriamo un grafo fissato, suff. grande  $G = G_n$

Fissiamo un qualunque nodo  $s \in V$ , ed eseguiamo una visita BFS partendo da  $s$ .

Dato che  $G$  è  $\alpha$ -expander, possiamo definire l'insieme

$$L_t = \{v \in V : d(s, v) = t\}, t = 0, 1, \dots, n-1$$

Osserviamo quindi che  $L_0 = \{s\}$  e  $L_1 = N(s) \geq \alpha$ , e quindi  $L_1 \geq 1$  dato che  $|N(s)|$  è intero.

Ora definiamo le seguenti famiglie di sottoinsiemi

$$I_0 = L_0; \quad I_t = I_{t-1} \cup L_t, \quad t = 0, 1, \dots$$

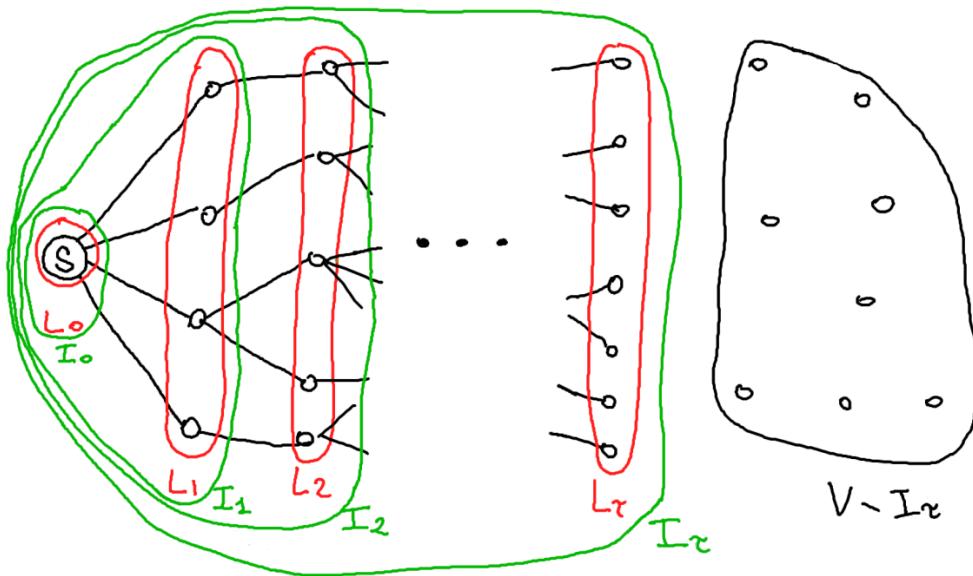
Notiamo che, per costruzione, vale che  $N(I_{t-1}) = L_t$  e che quindi  $|I_t| = |I_{t-1}| + |L_t|$

Ora, dato che  $G$  è  $\alpha$ -expander, fintanto che  $|I_{t-1}| \leq \frac{n}{2}$ , noi otteniamo che

$$|I_t| = |I_{t-1}| + |L_t| \geq (1 + \alpha)|I_{t-1}| \geq (1 + \alpha)^2|I_{t-2}| \geq \dots \geq (1 + \alpha)^{t-1}$$

Sia ora  $L_\tau$  un livello tale che il numero di nodi che appartengono al livello al più  $\tau$  è maggiore della metà dei nodi, quindi

$$\tau = \min \left\{ t \geq 1 : |I_t| > \frac{n}{2} \right\}$$



Allora  $\tau$  indica la distanza entro la quale almeno la metà dei nodi del grafo sono distanti da  $s$ . Otteniamo quindi che

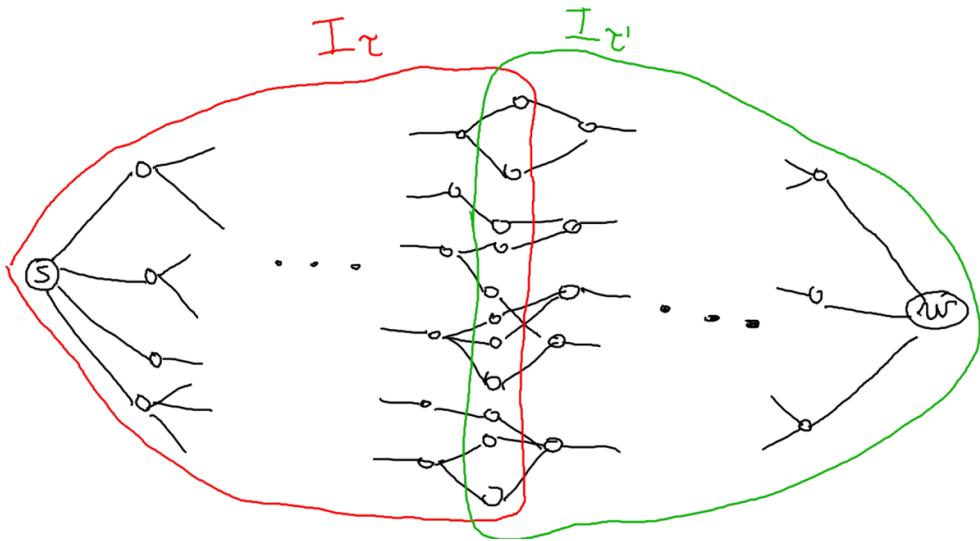
$$|I_\tau| \geq (1 + \alpha)^{\tau-1} = \frac{n}{2} \iff \tau = \log_{1+\alpha} \left( \frac{n}{2} \right) + 1$$

Quindi il numero di nodi a distanza  $\tau = O(\log(n))$  da  $s$  sono almeno  $\frac{n}{2}$  (ovvero  $|I_\tau| \geq \frac{n}{2}$ )

Ora consideriamo un qualunque altro nodo  $w \in V \setminus I_\tau$  e ripetiamo il medesimo procedimento con la BFS, partendo da  $w$ .

Anche qui, grazie alla proprietà expander di  $G$ , dopo  $\tau_1 = O(\log(n))$  livelli dell'albero BFS radicato in  $w$ , otteniamo che il sottoinsieme corrispondente  $I_{\tau_1}$  ha raggiunto dimensione almeno  $\frac{n}{2}$

Dato che entrambe le visite (quella che parte da  $s$  e quella che parte da  $w$ ) raggiungono entrambe *almeno* la metà dei nodi, significa che i due alberi costruiti dalle BFS devono condividere *almeno* un nodo



Allora data una qualsiasi coppia di nodi  $u, v \in V$  esiste sempre un cammino di lunghezza  $O(\log n)$  tra  $u$  e  $v$ .

Ne segue quindi che il diametro di  $G$  è  $O(\log n)$ , ■

Dalla dimostrazione del teorema segue il seguente corollario :

### ☒ Corollario 4.3 >

Consideriamo una famiglia di grafi infinita di dimensione crescente, ovvero

$$\{G_n(V_n, E_n) : |V_n| = [n], n \geq 1\}$$

Se esiste una costante assoluta  $\alpha > 0$  tale che, per un  $n$  suff. grande, il grafo  $G_n$  è un  $\alpha$ -expander allora il **tempo di completamento** del protocollo FLOOD è pari a  $O(\log(n))$

# Lezione 19 ADRC - Protocollo PULL su DELTA-Regular Expanders, Majority Consensus

## Protocollo PULL su $\Delta$ -regular Expanders

Consideriamo il seguente protocollo di comunicazione sparso su un grafo  $\Delta$ -regular, con  $\Delta \geq 2$ , partendo da una sorgente fissata  $s \in V$  avente un pezzo dell'informazione. Il goal è quello di far sì che tutti i nodi di  $V$  siano informati su  $s$ .

### Protocollo Randomizzato PULL( $G(V,E);s$ )

- Fase 0 (Wake-Up) : Il nodo informato è solo  $s$ , mentre tutti gli altri sono in stato ACTIVE.
- Fase 1 : Ogni nodo in stato ACTIVE sceglie u.a.r uno dei suoi vicini
  - Allora, se  $v$  è nello stato INFORMED, il nodo  $v$  fa il PULL del messaggio e si mette nello stato INFORMED.
  - I nodi nello stato INFORMED non fanno nulla.

Vale quindi il seguente teorema :

#### Teorema 5.1 >

Sia  $G = (V, E)$  un grafo  $\Delta$ -regolare con espansione  $\lambda$ , dove  $\lambda > 0$  è una costante positiva. Allora, partendo da ogni sorgente  $s \in V$ , il protocollo PULL informa tutti i nodi di  $V$  entro  $O(\log \Delta)$  tempo w.h.p.

#### dim

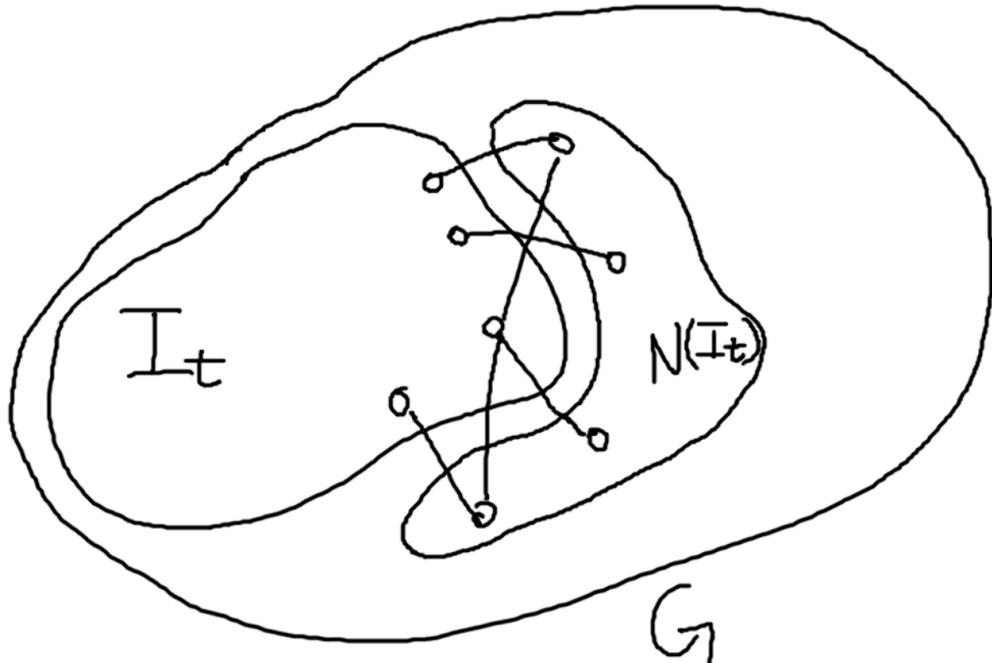
Per semplicità, eseguiremo solo l'analisi in aspettazione.

Definiamo le v.a

Il nostro prossimo obiettivo è dimostrare che la dimensione attesa di  $|F_t|$  aumenta **esponenzialmente** con  $t$ .

Per farlo sfruttiamo l'**espansione** di  $G$  e la **casualità** del protocollo PULL.

In dettaglio, si fissi un qualsiasi  $s \in V$  e si consideri il sottoinsieme di nodi nella frontiera di  $s$ , cioè



Definiamo inoltre la v.a

Allora, per ogni  $\alpha \in \mathbb{R}$ , vale che

$$\text{---} \leq \text{---}$$

D'altra parte, possiamo fornire un **lower bound** al numero di nodi in  $N(I_t)$  usando le proprietà di espansione del grafo  $G$ .

Infatti, fintanto che  $\alpha < 1 - \frac{1}{d}$  sappiamo che

$$\text{---} \leq \text{---}$$

Usando le ultime due diseguaglianze, possiamo dare un lower bound al **numero atteso di nuovi nodi informati al round  $t+1$** , come segue :

$$\text{---} \leq \text{---}$$

La diseguaglianza di cui sopra mostra che, fintanto che  $\alpha < 1 - \frac{1}{d}$ , la sua dimensione **ATTESA** incrementa di un fattore costante, dato che  $G$  è  $d$ -regular con

Quindi, dopo  $t+1$  rounds, almeno  $\alpha^t n$  nodi saranno informati, in **media**

$$\text{---} \leq \text{---}$$

Per concludere al  $\alpha = \frac{1}{d}$  la dimostrazione, dovremmo dimostrare che il numero di nodi informati passa da  $n$  a  $\alpha^n n$  in un numero logaritmico di round, ma questa parte viene omessa.

# Majority Consensus via the $\epsilon$ -Majority Dynamics

Il problema di trovare un *agreement* tra i nodi di una rete distribuita è uno dei problemi più importanti nei sistemi distribuiti moderni.

Qui studieremo questo problema, noto con il nome di **Majority Consensus Problem**, e vedremo un protocollo per tale problema, chiamato  $\epsilon$ -MAJ

Vediamo il problema in modo formale.

Sia  $G = (V, E)$  un grafo,  $C \subseteq \{0, 1\}^V$  l'insieme binario di colori e sia  $\pi_0 \in C$  un coloramento iniziale dei nodi di  $V$ .

Il goal algoritmico qui è progettare un protocollo semplice ed efficiente per il *Majority Consensus*.

In questo task, assumiamo che il coloramento iniziale abbia un certo *bias*  $\epsilon$  verso un qualche colore di maggioranza. Il goal è far sì che il sistema **converga** ad una configurazione **monocromatica** dove tutti i nodi ottengono il colore di maggioranza.

Chiariamo il concetto di bias di una configurazione.

Data una configurazione  $\pi$ , per ogni colore  $c \in \{0, 1\}$  definiamo  $\pi_c$  come la dimensione del  $c$ -esimo colore, ovvero il numero di nodi aventi colore  $c$ .

Assumendo che  $\pi \neq \pi_0$ , il bias di  $\pi$  è definito come

$$\epsilon = \frac{\pi_1 - \pi_0}{\pi_1 + \pi_0}$$

Definiamo ora, in modo formale, il problema del  $\epsilon$ -Majority-Consensus :

## $\epsilon$ -Majority-Consensus >

Dato un sistema distribuito  $(G, \pi_0)$ , il problema  $\epsilon$ -Majority-Consensus è definito dalla seguente proprietà.

Partendo da un qualunque coloramento iniziale  $\pi_0$  avente bias  $\epsilon$ , il sistema **deve convergere** a una configurazione stabile in cui tutti i nodi **supportano** (hanno) il colore di maggioranza.

Definiamo ora il protocollo  $\epsilon$ -MAJ

## Protocollo $\epsilon$ -MAJ

- Ad ogni round  $t$ , ogni nodo sceglie indipendentemente i nodi vicini (incluso se stesso e con ripetizioni) u.a.r e ricolora se stesso in accordo alla maggioranza calcolata tra i nodi scelti.

Possiamo dimostrare che  $\text{-MAJ}$  non produce alcuna deriva verso il colore di maggioranza, non importa quale sia il bias attuale del sistema: tradotto, il bias non aumenta **in media!**

Questo fatto ha due conseguenze principali.

In primo luogo vale la regola che

---

Questo implica che, anche partendo da un bias  $\epsilon$ , la probabilità di errore delle due dinamiche è ancora maggiore di una costante assoluta.

La seconda cattiva notizia è che le dinamiche di cui sopra sono molto lente a convergere: richiedono un numero **polinomiale** di passi!

Questo fatto è piuttosto difficile da dimostrare ma è essenzialmente dovuto al fatto che, come osservato in precedenza, la dinamica non ha una deriva attesa e la sua convergenza è dovuta solo alla casualità (imprevedibile) del processo.

Per i fatti di cui sopra, ci concentreremo sul  $\text{-MAJ}$ .

## Unbalanced -coloring with -MAJ

Analizziamo il  $\text{-MAJ}$  nel caso del  $\text{-coloramento}$ , con

### Definizione 6.3 >

Per un  $\text{-coloramento}$   $\{C_0, C_1\}$ , diciamo che  $C_0$  è  $\epsilon$ -sbilanciato se il suo bias è tale da avere

Nel prossimo lemma dimostreremo che, se la config. iniziale è sufficientemente sbilanciata, allora il  $\text{-MAJ}$  risolve il problema del Majority Consensus entro  $O(\log n / \epsilon^2)$  round w.h.p

### Lemma 6.4 >

Se  $\epsilon < \frac{1}{2}$  e il  $\text{-coloramento}$  iniziale è  $\epsilon$ -sbilanciato, allora il  $\text{-MAJ}$  converge al colore di maggioranza dopo  $O(\log n / \epsilon^2)$  round w.h.p

**dim**

Sia  $R_t$  la v.a. che conta il numero di nodi **rossi** al tempo  $t$ .

Per ogni nodo  $v$  si definisca la v.a. indicatrice dell'evento "nodo  $v$  è **rosso** al prossimo step".

Per ogni vale che

$$\boxed{- \quad - \quad -}$$

Quindi, il numero atteso di nodi colorati di **rosso** al prossimo time step è :

$$\boxed{-}$$

W.l.o.g assumiamo che il **rosso** sia il colore di minoranza. Dividiamo l'analisi in tre fasi in accordo al range i minoranza in cui cade il valore

### **Analisi : Fase 1 - sta nel range da $\boxed{a}$**

Supponiamo che il numero di nodi colorati di rosso sia  $\boxed{\quad}$  per qualche  $\boxed{\quad}$ , dove per una qualche costante positiva  $\boxed{\quad}$ .

Dimostreremo che  $\boxed{\quad}$  w.h.p

Osserviamo che la funzione

è crescente per ogni

Quindi, per ogni valore  $\boxed{\quad}$  abbiamo che :

$$\boxed{- \quad - \quad - \quad - \quad -}$$

dove l'ultima disegualianza vale perchè

Ora andrebbe applicato il Chernoff bound (forma additiva) dato che le  $\boxed{\quad}$  sono tutte indipendenti condizionate a  $\boxed{\quad}$ , usando come variabili

$$\boxed{- \quad - \quad -}$$

ma questa parte viene omessa.

### **Analisi : Fase 2 - sta nel range da $\boxed{a}$**

Se  $\boxed{\quad}$  con  $\boxed{\quad}$ , da  $\boxed{\quad}$  otteniamo

Anche qui applichiamo il Chernoff Bound (forma moltiplicativa) con

(anche qui però è omesso)

Quindi fintanto che  $\frac{1}{n} \sum_{i=1}^n p_i \geq \frac{1}{2}$  allora il numero di nodi **rossi** **decresce esponenzialmente** w.h.p.

Ragionando come nella fase precedente otteniamo che dopo ulteriori  $t$  time steps il numero di nodi **rossi** sarà

### Analisi : Fase 3 - sta nel range da $a$

Osserviamo che  $\frac{1}{n} \sum_{i=1}^n p_i \geq \frac{1}{2}$ , in eq. (1) otteniamo che

per una qualunque costante positiva  $c > 0$ .

Quindi, usando la Markov Inequality con

otteniamo che

e poiché  $\frac{1}{n} \sum_{i=1}^n p_i$  è a valori interi ne consegue che tutti i nodi sono colorati di **blu** w.h.p.