Filtering: A Method for Solving Graph Problems in MapReduce

S. Lattanzi B. Moseley S. Suri S. Vassilvitskii

Giulio Ardito

Sommario

Filtering

- Cos'è?

Filtering

- Cos'è?
- A quale modello fa riferimento?

Filtering

- Cos'è?
- A quale modello fa riferimento?
- Quali problemi cerca di risolvere?

Filtering

Idea alla base del Filtering

Ridurre la grandezza dell'input con *MapReduce*, in modo tale che il problema, con dimensione molto più piccola, possa essere risolto su una singola macchina.

Filtering

Idea alla base del Filtering

Ridurre la grandezza dell'input con *MapReduce*, in modo tale che il problema, con dimensione molto più piccola, possa essere risolto su una singola macchina.

Dall'idea alla pratica

Nei problemi su grafi, non tutti gli archi sono utili ai fini della soluzione!

Riassumendo

- "Filtraggio" dell'input tramite MapReduce
- Soluzione del problema su una singola macchina

Problema

Come scegliere gli archi da scartare?

Obiettivi e restrizioni

Obiettivi:

- Mapper e Reducer lavorano su sottografi

Obiettivi e restrizioni

Obiettivi:

- Mapper e Reducer lavorano su sottografi
- Mantenere tempo polinomiale in ogni round

Obiettivi e restrizioni

Obiettivi:

- Mapper e Reducer lavorano su sottografi
- Mantenere tempo polinomiale in ogni round
- Tempo limitato dal numero di round

Obiettivi e restrizioni

Obiettivi:

- Mapper e Reducer lavorano su sottografi
- Mantenere tempo polinomiale in ogni round
- Tempo limitato dal numero di round

Restrizioni:

Obiettivi e restrizioni

Obiettivi:

- Mapper e Reducer lavorano su sottografi
- Mantenere tempo polinomiale in ogni round
- Tempo limitato dal numero di round

Restrizioni:

- Nessuna macchina può vedere l'intero input

Obiettivi e restrizioni

Obiettivi:

- Mapper e Reducer lavorano su sottografi
- Mantenere tempo polinomiale in ogni round
- Tempo limitato dal numero di round

Restrizioni:

- Nessuna macchina può vedere l'intero input
- Comunicazioni fra macchine non consentita

Map Reduce Class ($\mathcal{MRC}$)

[Karloff, Suri and Vassilvitskii]

- $N =$ input size
- $\epsilon > 0$ costante fissata sufficientemente piccola
- $N^{1-\epsilon}$ macchine
- $N^{1-\epsilon}$ memoria su ogni macchina
- Memoria disponibile sull'intero sistema $O(N^{2-2\epsilon})$

$\mathcal{MRC}^i$: problemi che possono essere risolti in $O(\log^i N)$ rounds.

Total work e work efficiency

Total work

$w(N) = \sum_{i=1}^{r} w_i(N) = \sum_{i=1}^{r} p_i(N) t_i(N)$

Work efficiency

Se il lavoro svolto da un algoritmo $\mathcal{MRC}$ raggiunge il tempo di esecuzione del miglior algoritmo sequenziale conosciuto, allora l'algoritmo è "work efficient".

Definizioni

- $G=(V,E)$: grafo in input
- n : numero di nodi
- m : numero di archi
- η : memoria disponibile su ogni macchina
- N : dimensione dell'input

Assunzioni

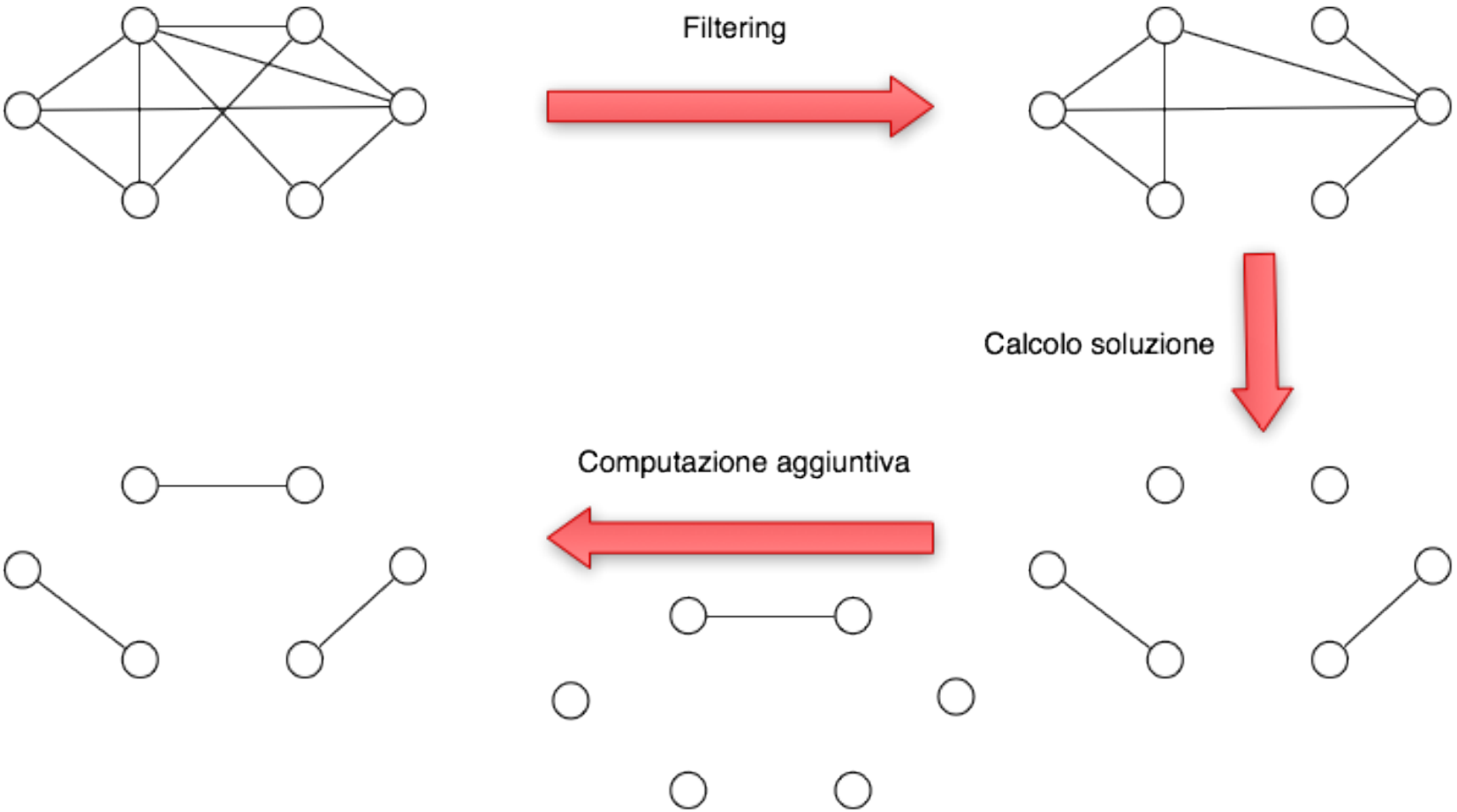
- Numero macchine: $O(m/\eta)$
- Grafi densi: $m = n^{1+c}$, con $0 < c \leq 1$
- Studi dimostrano che i social networks sono grafi *densi*

Perché grafi densi?

[Leskovec, Kleinberg and Faloutsos]

- Studio su 9 diversi social network
- Vari grafi aventi n^{1+c} archi
- Più piccolo valore di $c = 0.08$
- $c \geq 0.5$ su quattro grafi

Warm-up



Minimum Spanning Tree

Sia $G = (V, E)$ un grafo connesso non orientato e $w : E \to R$ una funzione costo degli archi di G. Definiamo inoltre $m : V \to V$ nel seguente modo: m(u)=v sse (u,v) è l'arco di costo minimo incidente su u.

Un MST di $G = (V, E)$ è un albero ricoprente $T = (V, E')$ di costo minimo ($E' \subseteq E$).
Il costo di un albero è la somma dei costi degli archi che lo compongono: $w(T) = \sum_{e \in T} w(e)$.

Algoritmi noti

Prim

Si parte da T = un singolo vertice e si costruisce incrementalmente il MST aggiungendo l'arco di costo minimo tra T e G-T.

Kruskal

Si parte da una foresta di nodi isolati e, considerando tutti gli archi in ordine di costo crescente, si aggiunge ciascun arco solo se non induce un ciclo.

Sollin

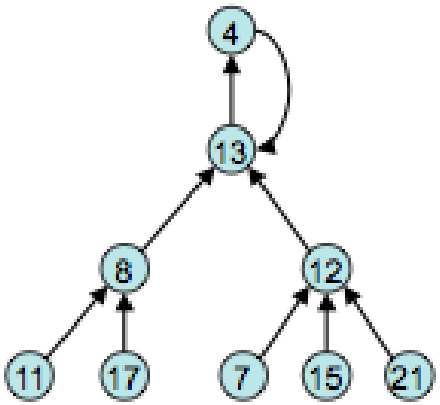
Si parte da una foresta di nodi isolati, si aggiungono tutti gli archi (u,m(u)) e si itera sugli archi che uniscono le varie componenti connesse fino ad ottenere un albero.

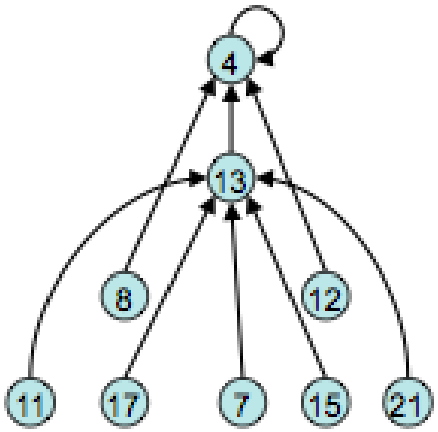
MST Parallelo

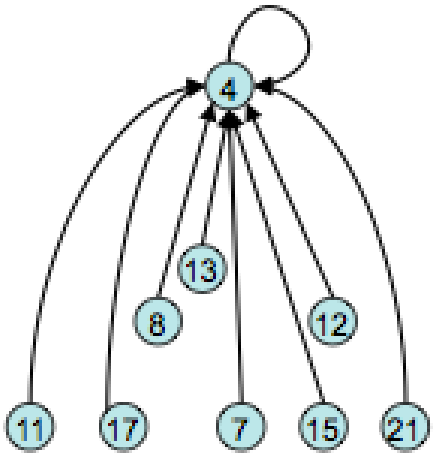
Si basa su:

- Algoritmo di Sollin
- Alberi radicati e stelle radicate
- Pseudo-foreste e meganodi

MST Parallelo

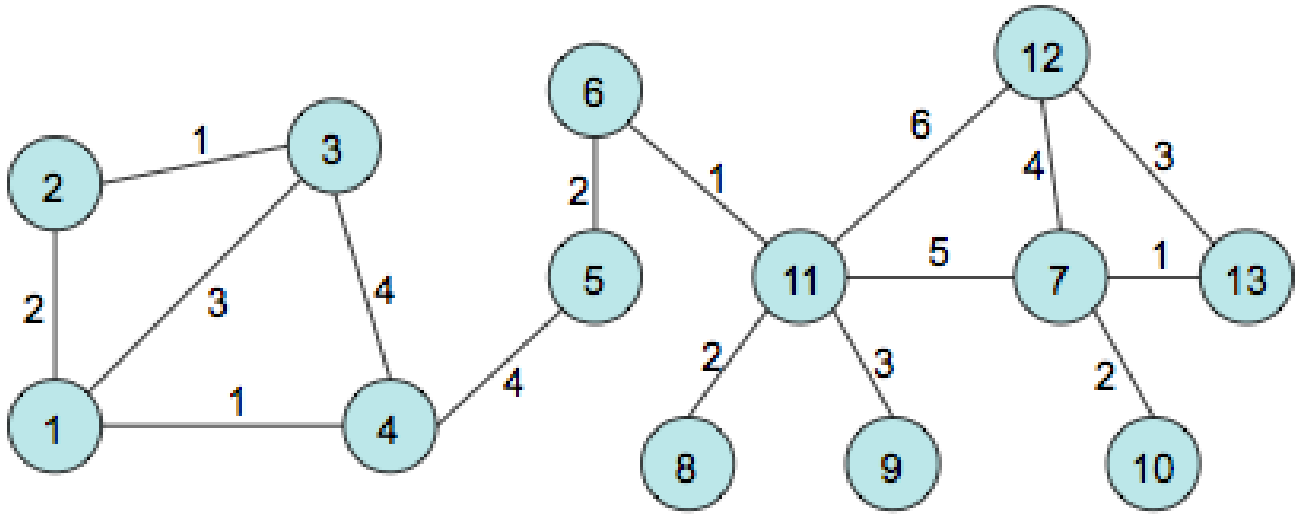




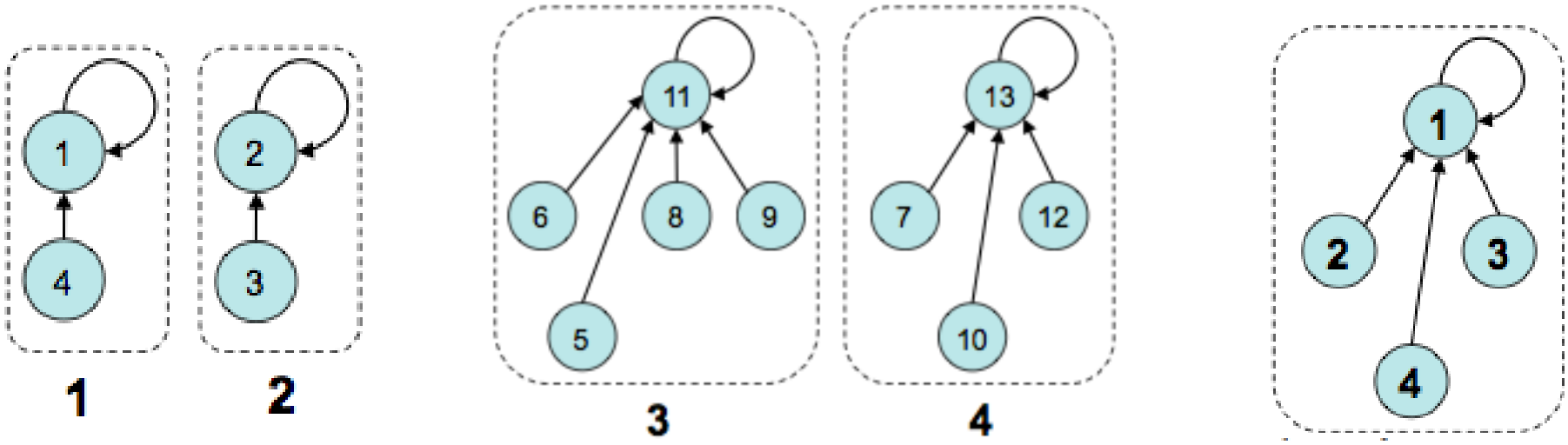


MST Parallelo

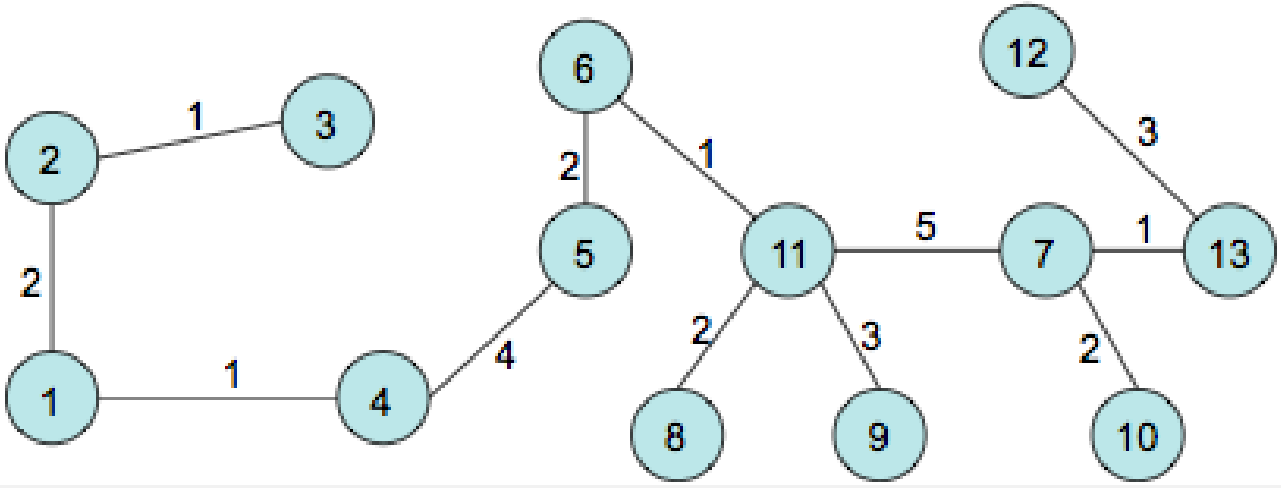
Come funziona:



MST Parallelo



MST Parallelo



MST Parallelo

Costo dell'algoritmo:

- Ricerca minimo adiacente: $O(\log n)$ su EREW, $O(1)$ su ERCW
- Salto del puntatore: $O(\log n)$ su CREW
- Somme prefisse (numerazione meganodi): $O(\log n)$ su EREW
- Generazione matrice adiacenza: $O(\log n)$ su EREW

MST Parallelo

Costo dell'algoritmo:

- Ricerca minimo adiacente: $O(\log n)$ su EREW, $O(1)$ su ERCW
- Salto del puntatore: $O(\log n)$ su CREW
- Somme prefisse (numerazione meganodi): $O(\log n)$ su EREW
- Generazione matrice adiacenza: $O(\log n)$ su EREW

L'algoritmo richiede $O(\log^2 n)$ tempo su una PRAM CREW con n^2 processori.

MST Distribuito

Si basa su:

- Algoritmo di Sollin
- Se tutti gli archi di un grafo G hanno pesi differenti, allora esiste un unico albero di copertura di G di costo minimo.
- Dato un sottoalbero T' di un qualche MST T di costo minimo e un arco e uscente da T' di costo minimo, allora $T' \cup \{e\}$ è ancora un sottoalbero di qualche albero di copertura di costo minimo (unicità dei pesi non necessaria).

MST Distribuito

Panoramica

1. Broadcast con eco

MST Distribuito

Panoramica

1. Broadcast con eco
2. Costruzione di frammenti

MST Distribuito

Panoramica

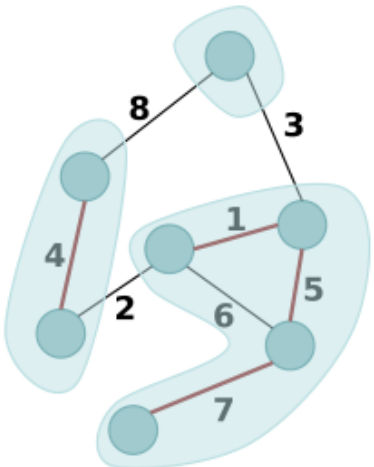
1. Broadcast con eco
2. Costruzione di frammenti
3. Ricerca arco uscente di costo minimo dal frammento

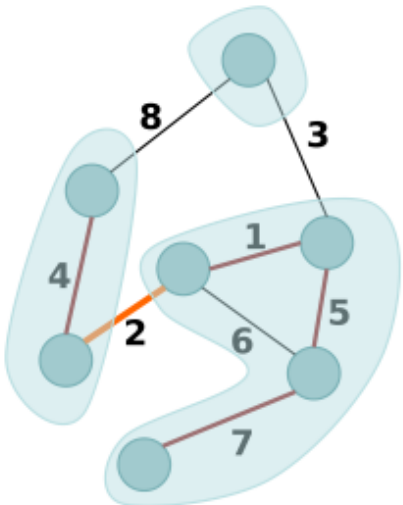
MST Distribuito

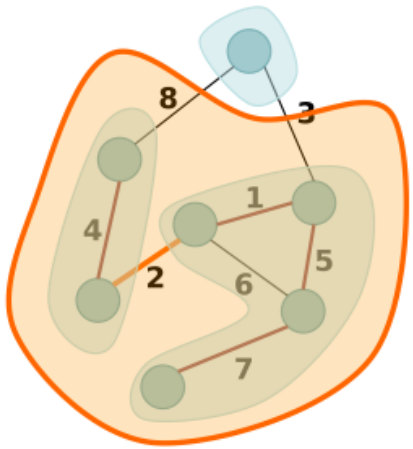
Panoramica

1. Broadcast con eco
2. Costruzione di frammenti
3. Ricerca arco uscente di costo minimo dal frammento
4. Procedura di aggregazione

MST Distribuito







MST Distribuito

Costo dell'algoritmo:

- Complessità messaggi: $O(n \log n + m)$
- Complessità temporale: $O(n \log n)$

MST MapReduce

Elementi in comune:

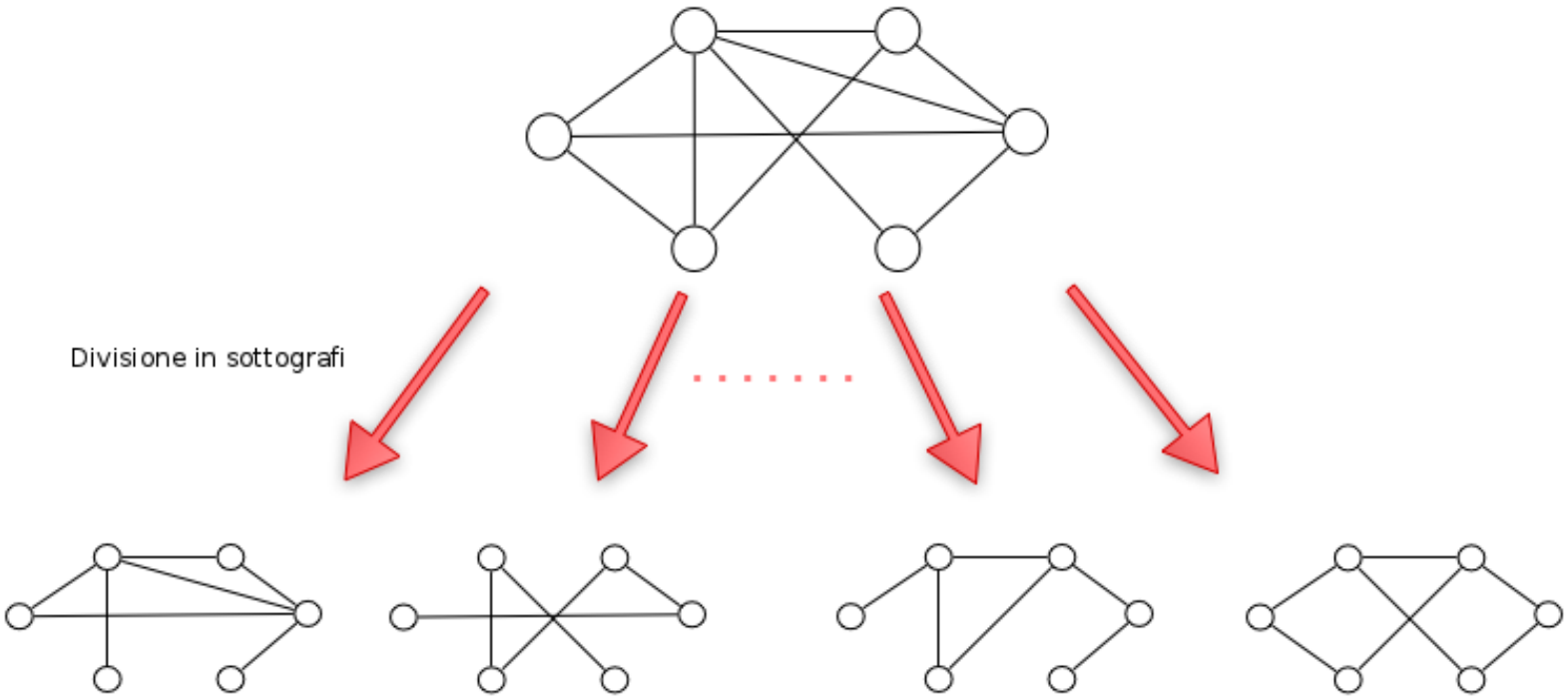
- Algoritmo di Sollin (in particolare MST di Chazelle)
- Componenti connesse
- Utilizzo di frammenti
- Proprietà MST
- Scarto archi

MST MapReduce

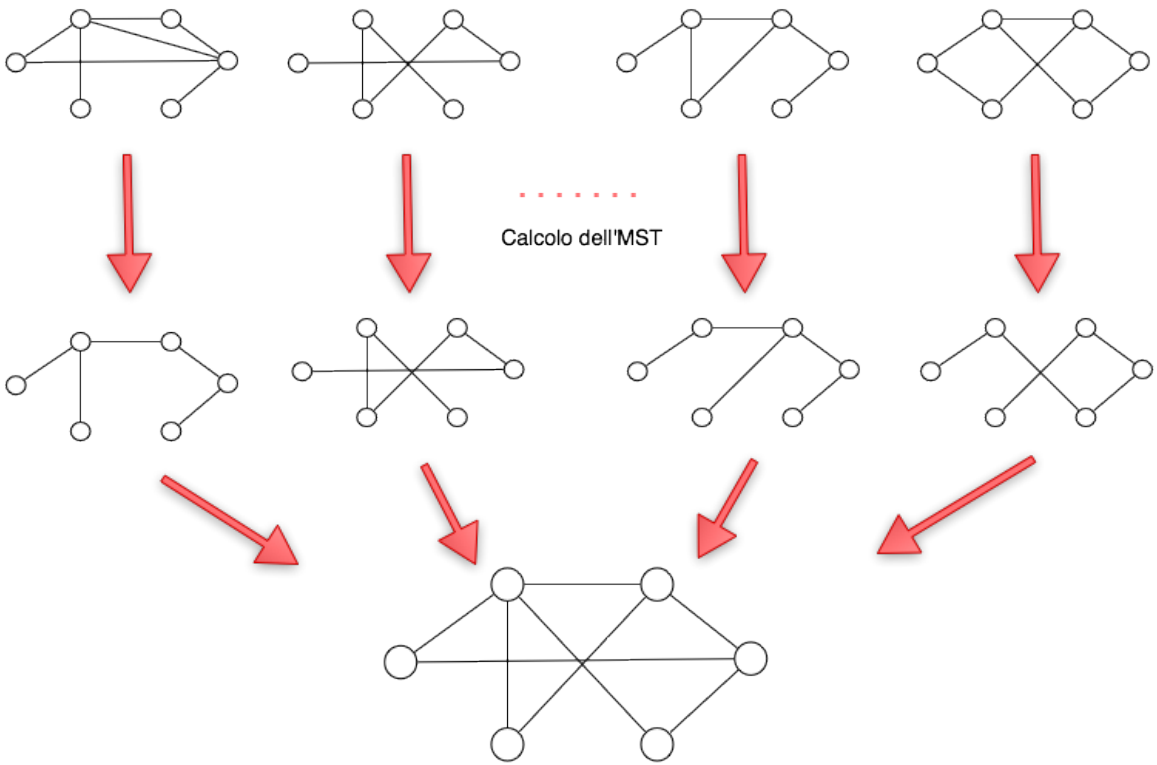
Algorithm: MST(V,E)

1: **if** $|E| < \eta$ **then**
2: Compute $T^* = MST(E)$
3: **return** T^*
4: **end if**
5: $\ell \leftarrow \Theta(|E|/\eta)$
6: Partition E into $E_1, E_2, \ldots, E_\ell$ where $|E_i| < \eta$ using a universal hash function $h : E \rightarrow \{1, 2, \ldots, \ell\}$.
7: In parallel: Compute T_i, the minimum spanning tree on $G(V, E_i)$.
8: **return** MST$(V, \cup_i T_i)$

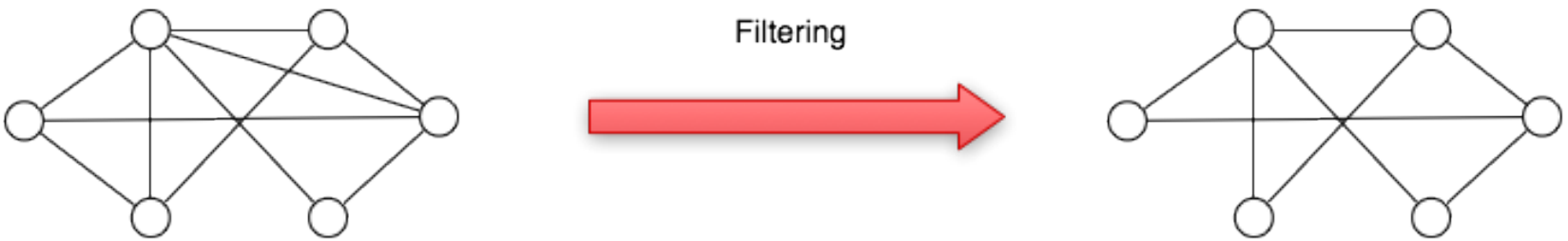
MST MapReduce



MST MapReduce



MST MapReduce



Considerazioni

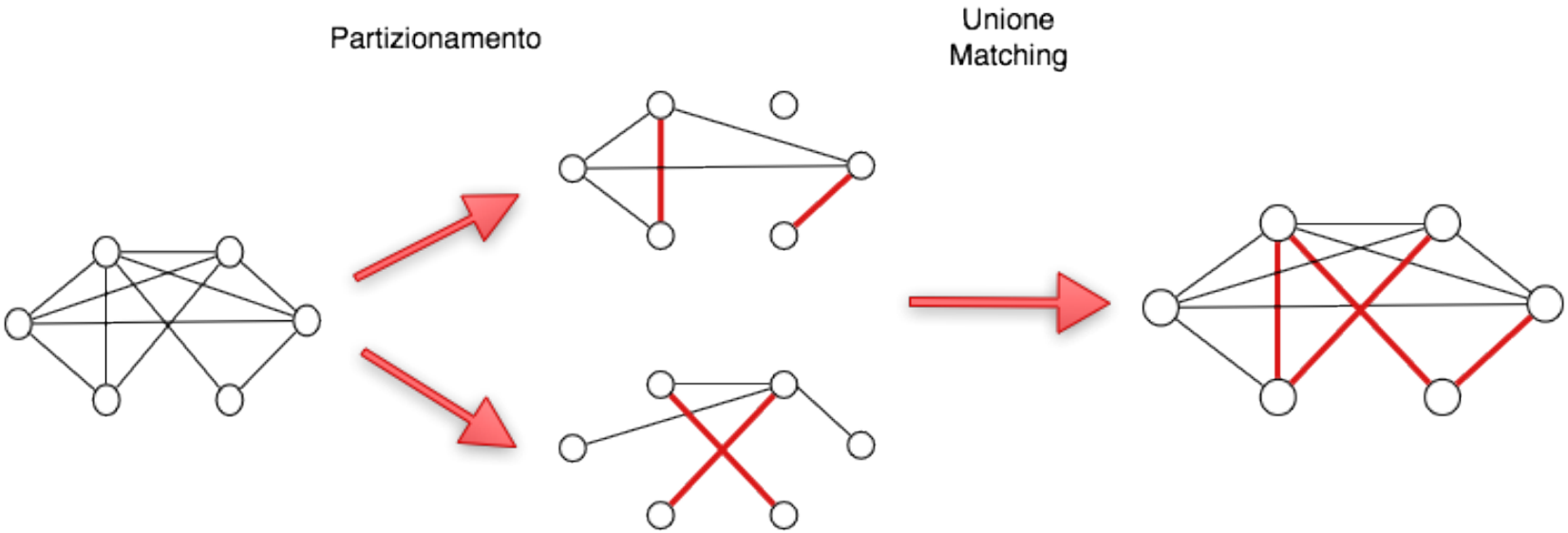
- MST su Parallelo/Distribuito/MapReduce
- Nessun arco nella soluzione finale viene scartato nella soluzione parziale
- $O(n^{c-\epsilon})$ macchine usate
- $O(n^{1+\epsilon})$ memoria utilizzata da ogni macchina
- L'algoritmo termina dopo $\lceil c/\epsilon \rceil$ rounds

Unweighted Maximal Matching

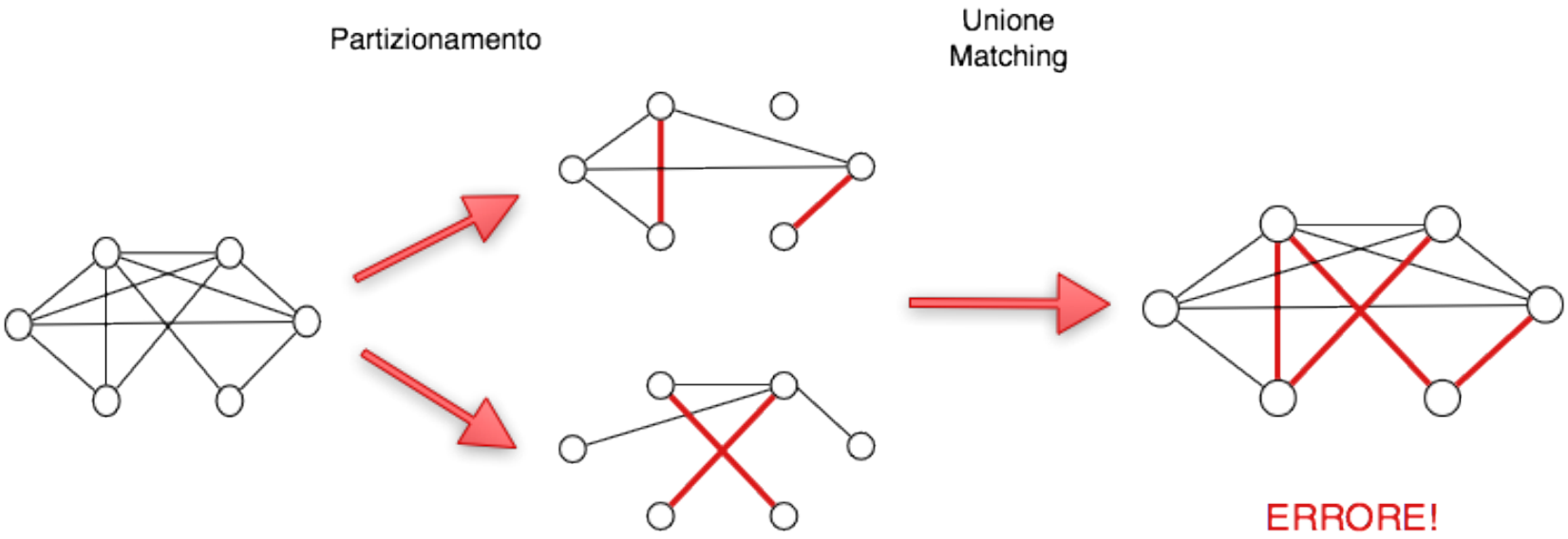
Idea

con la stessa intuizione del calcolo dell'MST approcciamo al calcolo di matching massimali su grafi non pesati.

Primo approccio



Primo approccio



Pseudocodice

1. Set $M = \emptyset$ and $\mathcal{S} = E$.

2. Sample every edge $(u, v) \in \mathcal{S}$ uniformly at random with probability $p = \frac{\eta}{10|\mathcal{S}|}$. Let E' be the set of sampled edges.

3. If $|E'| > \eta$ the algorithm fails. Otherwise give the graph $G(V, E')$ as input to a single machine and compute a maximal matching M' on it. Set $M = M \cup M'$.

4. Let I be the set of unmatched vertices in G. Compute the subgraph of G induced by I, $G[I]$, and let $E[I]$ be the set of edges in $G[I]$. If $|E_i| > \eta$, set $\mathcal{S} = E[I]$ and return to step 2. Otherwise continue to step 5.

5. Compute a maximal matching M'' on $G[I]$ and output $M = M \cup M''$

Linee guida

1. Si considera un qualsiasi $E' \subseteq E$

Linee guida

1. Si considera un qualsiasi $E' \subseteq E$
2. Si calcola il matching M' su $G[E']$

Linee guida

1. Si considera un qualsiasi $E' \subseteq E$
2. Si calcola il matching M' su $G[E']$
3. Si considera $I = \{v \in V' : v \notin M'\}$

Linee guida

1. Si considera un qualsiasi $E' \subseteq E$
2. Si calcola il matching M' su $G[E']$
3. Si considera $I = \{v \in V' : v \notin M'\}$
4. Si calcola il matching M'' su $G[I]$

Più nel dettaglio...

1. Campionamento degli archi con probabilità $p = \frac{\eta}{10|S|}$

Più nel dettaglio...

1. Campionamento degli archi con probabilità $p = \frac{\eta}{10|S|}$
2. Calcolo del matching M' sul grafo campionato $G[E']$

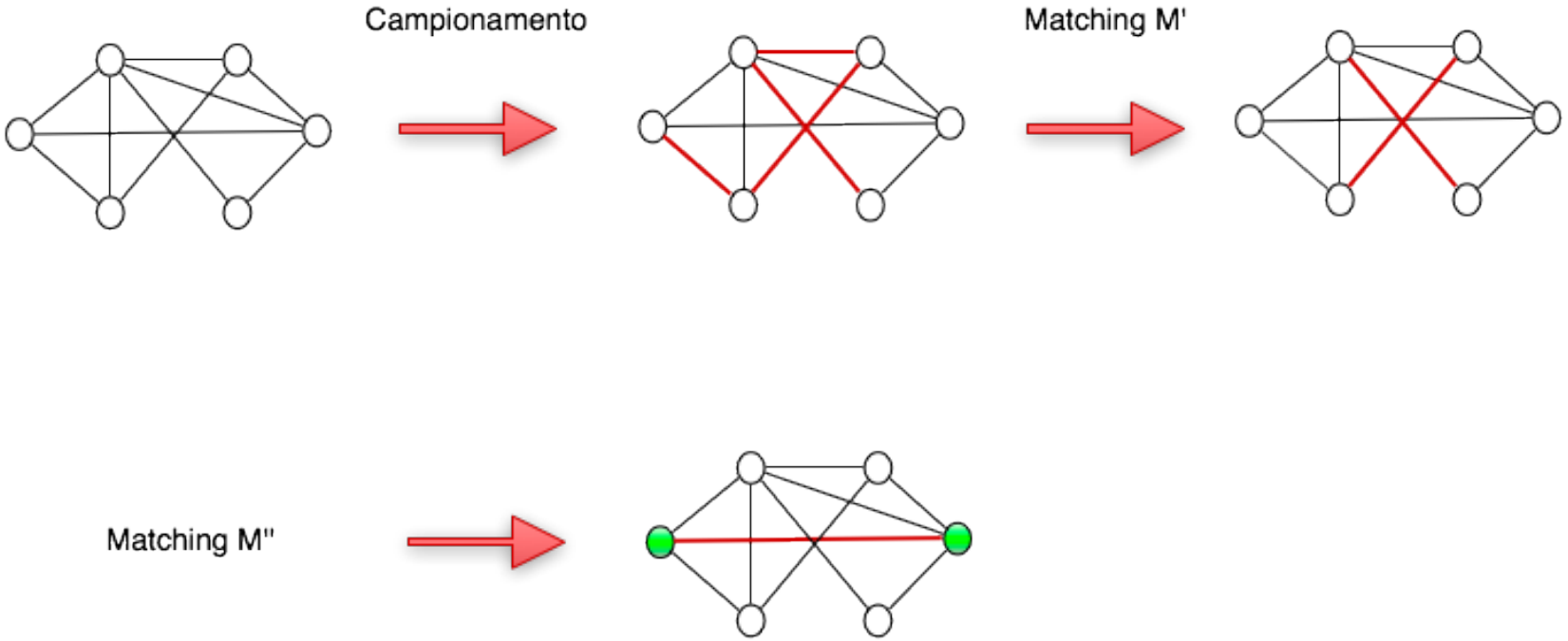
Più nel dettaglio...

1. Campionamento degli archi con probabilità $p = \frac{\eta}{10|S|}$
2. Calcolo del matching M' sul grafo campionato $G[E']$
3. Calcolo del matching M'' sul sottografo indotto dall'insieme $I = \{v \in V' : v \notin M'\}$

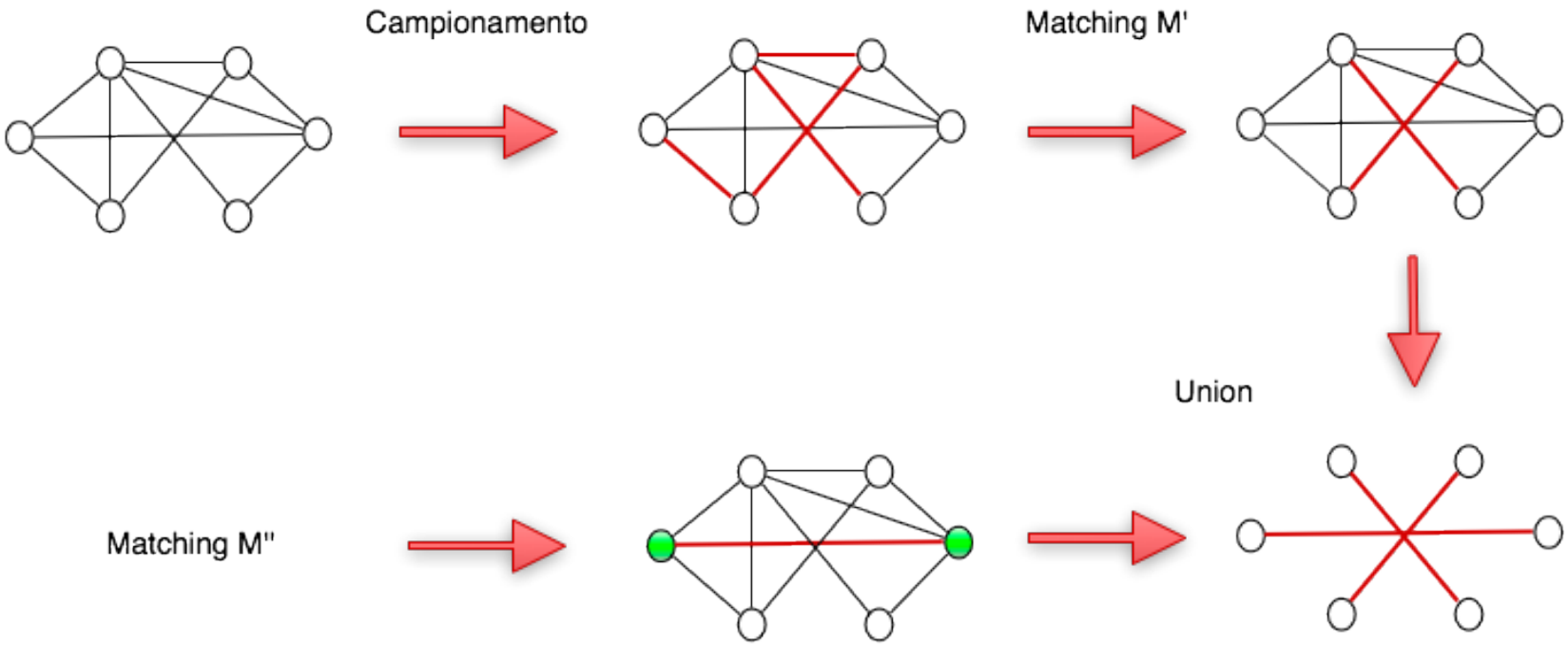
Più nel dettaglio...

1. Campionamento degli archi con probabilità $p = \frac{\eta}{10|S|}$
2. Calcolo del matching M' sul grafo campionato $G[E']$
3. Calcolo del matching M'' sul sottografo indotto dall'insieme $I = \{v \in V' : v \notin M'\}$
4. Return $M = M \cup M' \cup M''$

Esempio



Esempio



Considerazioni

L'algoritmo termina dopo:

- *3 rounds* se $\eta = n^{1+2c/3}$
- $3\lfloor c/\epsilon \rfloor$ *rounds* se $\eta = n^{1+\epsilon}$, con $0 < \epsilon < c$
- $O(\log n)$ rounds se $\eta \geq 40n$

Considerazioni

L'algoritmo termina dopo:

- *3 rounds* se $\eta = n^{1+2c/3}$
- $3\lfloor c/\epsilon \rfloor$ *rounds* se $\eta = n^{1+\epsilon}$, con $0 < \epsilon < c$
- $O(\log n)$ rounds se $\eta \geq 40n$

Cosa succede se $|E[I]| > \eta$?

Minimum Cut

Come filtrare gli archi del grafo?

- Partizionamento ci fa perdere informazioni strutturali
- Campionamento non funziona

Minimum Cut

Come filtrare gli archi del grafo?

- Partizionamento ci fa perdere informazioni strutturali
- Campionamento non funziona

Soluzione

Primo passo dell'algoritmo di Karger (algoritmo di contrazione)

Algoritmo di Karger

- Metodo Monte Carlo
- Algoritmo ricorsivo
- Basato sul concetto di contrazione di un arco
- Gli archi da contrarre sono scelti casualmente

Algoritmo di Karger

- Metodo Monte Carlo
- Algoritmo ricorsivo
- Basato sul concetto di contrazione di un arco
- Gli archi da contrarre sono scelti casualmente

Proprietà

Scelte fatte nei primi round hanno alta probabilità di successo.

Lattanzi et al.

- Archi etichettati con valori in [0,1]
- Rircerca di una soglia t
- Contrazione degli archi con etichetta minore di t
- Algoritmo di Min-Cut

Metodo Monte Carlo

1. Definisce un dominio di possibili dati in input
2. Genera input casuali dal dominio con una certa distribuzione di probabilità determinate
3. Esegue un calcolo deterministico utilizzando i dati in ingresso (input)
4. Aggrega i risultati dei calcoli singoli nel risultato finale

Pseudocodice

Algorithm 1 MinCut(E)

1: **for** $i = 1$ to n^{δ_1} (in parallel) **do**
2: tag $e \in E$ with a number r_e chosen uniformly at random from $[0, 1]$
3: $t \leftarrow \text{Find}_t(E, 0, 1)$
4: $E_i \leftarrow \text{Contract}(E, t)$
5: $C_i \leftarrow$ min cut of E_i
6: **end for**
7: **return** minimum cut over all C_i

Find e Contract

Algorithm 2 $\text{Find}_t(E, \min, \max)$

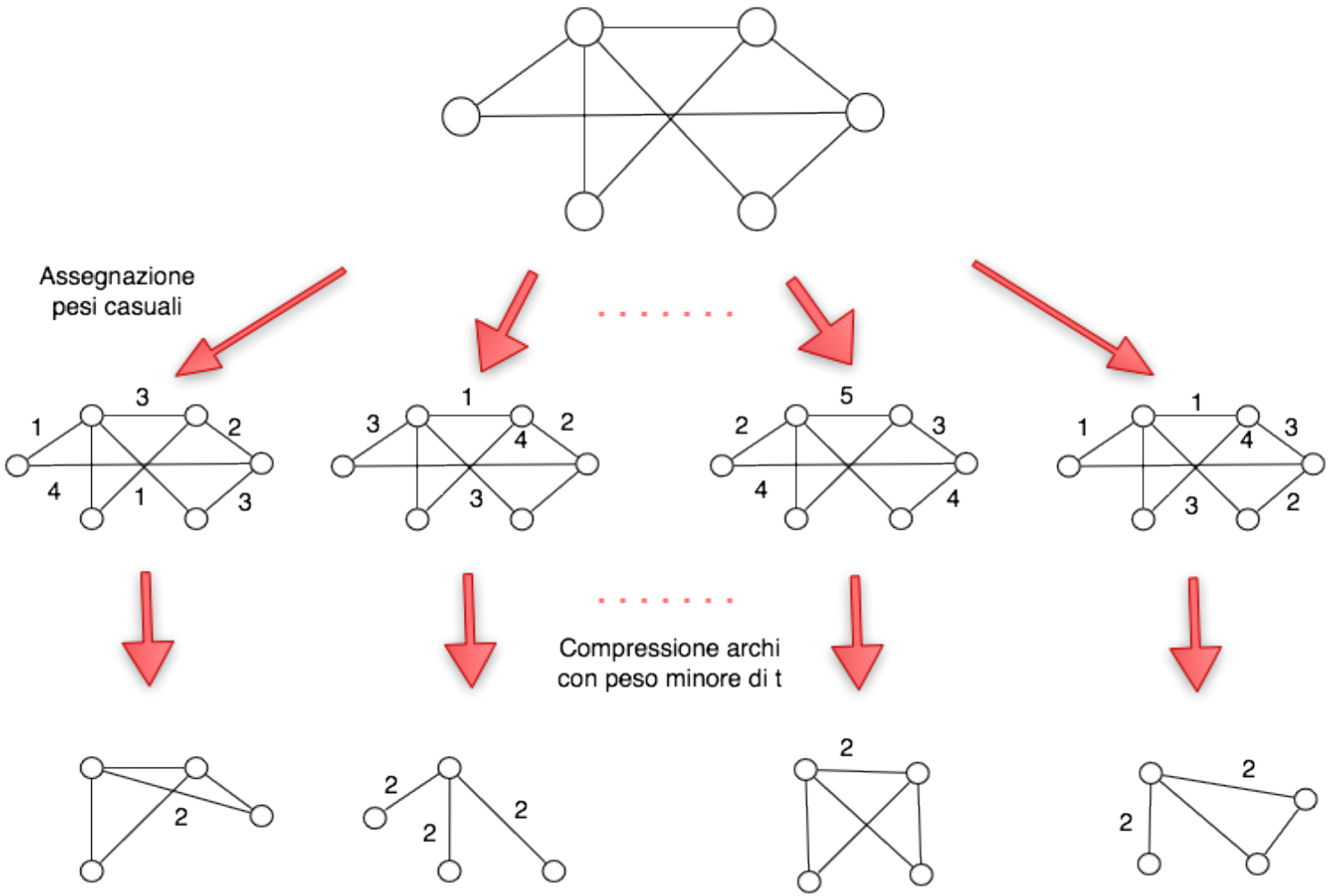
1: {Uses $n^{\delta_2 + c/\epsilon}$ machines.}
2: $\gamma \leftarrow \frac{\max - \min}{n^{\delta_2}}$
3: **for** $j = 1$ to n^{δ_2} (in parallel) **do**
4: $\quad \tau_j \leftarrow \min + j\gamma$
5: $\quad E_j \leftarrow \{e \in E \mid r_e \leq \tau_j\}$
6: $\quad cc_j \leftarrow$ number of connected components in $G = (V, E_j)$
7: **end for**
8: **if** there exists a j such that $cc_j = n^{\delta_3}$ **then**
9: $\quad$ **return** j
10: **else**
11: $\quad$ **return** $\text{Find}_t(E, \tau_j, \tau_{j+1})$ where j is the smallest value s.t. $cc_j < n^{\delta_3}, cc_{j+1} > n^{\delta_3}$
12: **end if**

Find e Contract

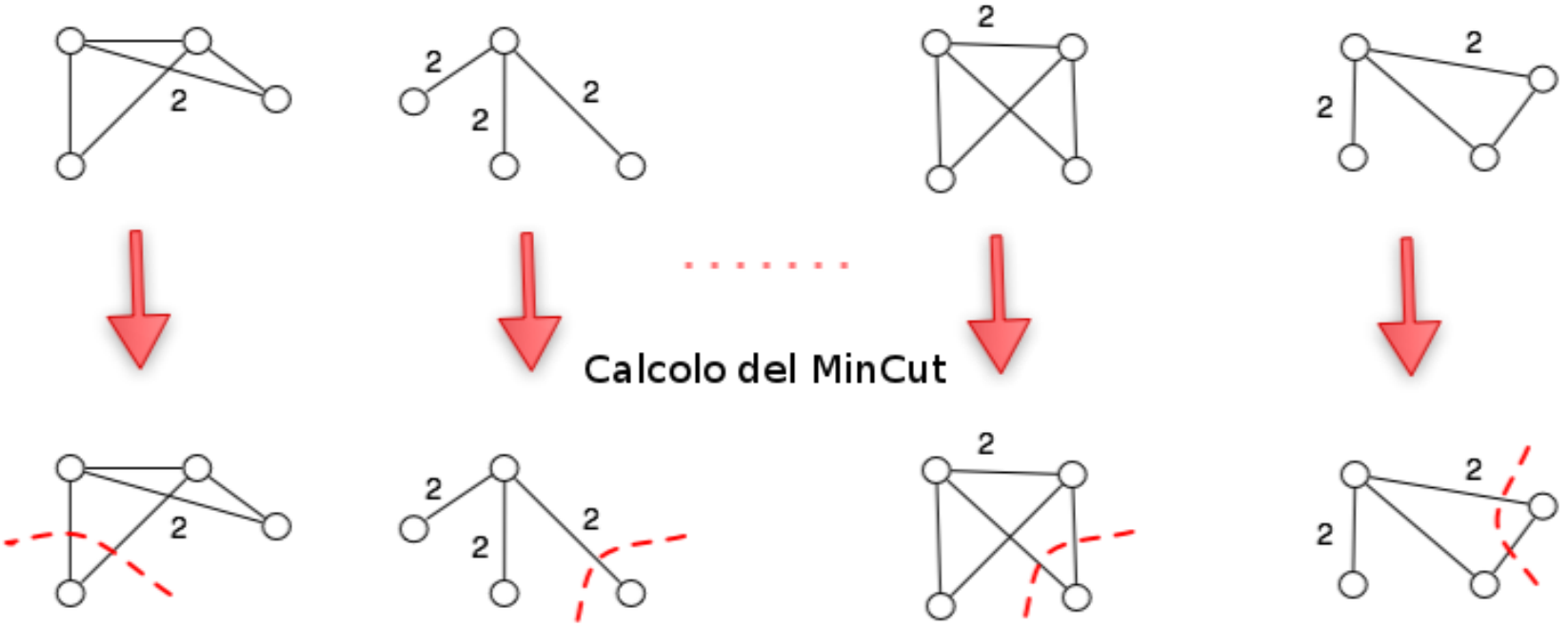
Algorithm 3 Contract(E, t)

1: $CC \leftarrow$ connected components in $\{e \in E \mid r_e \leq t\}$
2: let $h : [n] \rightarrow [n^{\delta_4}]$ be a universal hash function
3: map each edge (u, v) to machine $h(u)$ and $h(v)$
4: map the assignment of node u to its connected component $CC(u)$, to machine $h(u)$
5: on each reducer rename all instances of u to $CC(u)$
6: map each edge (u, v) to machine $h(u) + h(v)$
7: Drop self loops (edges in same connected component)
8: Aggregate parallel edges

Esempio


Assegnazione
pesi casuali
Compressione archi
con peso minore di t

Esempio



Considerazioni

- Memoria: $max\{n^{2\delta_3}, n^{1+c-\delta_4}, n^{1+\epsilon}\}$
- Macchine: $n^{\delta_1}(n^{\delta_2+c-\epsilon} + n^{\delta_4})$
- Rounds: $O(\frac{1}{\epsilon n^{\delta_2}})$
- Probabilità: $1 - (1 - \frac{n^{2\delta_3}}{n^2})^{n^{\delta_1}}$

Considerazioni

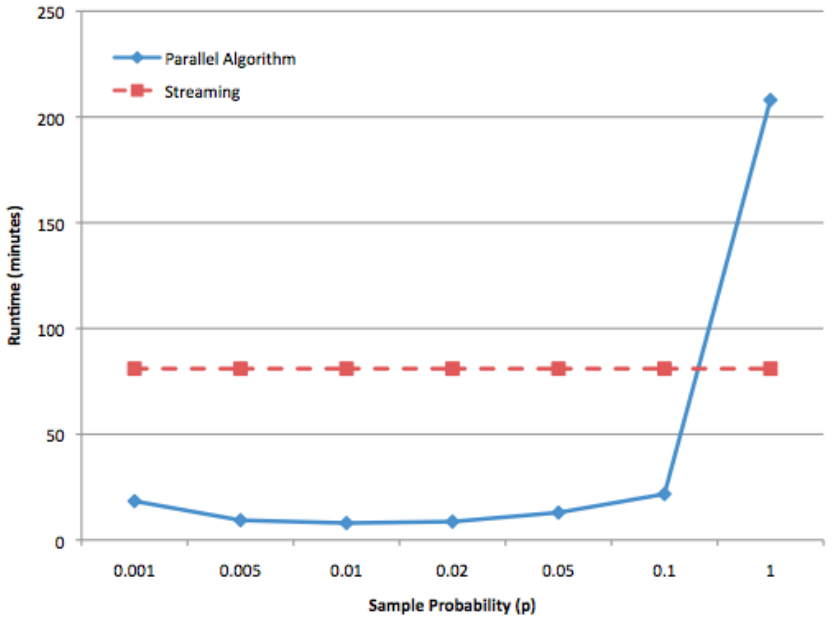
- $\delta_1 = 1 - \epsilon/2$
- $\delta_2 = \epsilon$
- $\delta_3 = \frac{1+\epsilon}{2}$
- $\delta_4 = c$

Considerazioni

- Memoria: $max\{n^{2\delta_3}, n^{1+c-\delta_4}, n^{1+\epsilon}\} \leq \eta = n^{1+\epsilon}$
- Macchine: $n^{\delta_1}(n^{\delta_2+c-\epsilon} + n^{\delta_4}) = o(m) = o(n^{1+c})$
- Rounds: $O(\frac{1}{\epsilon n^{\delta_2}}) = O(\frac{1}{\epsilon^2})$
- Probabilità: $1 - (1 - \frac{n^{2\delta_3}}{n^2})^{n^{\delta_1}} = 1 - o(1)$

Risultati

Matching: Greedy-Streaming vs MapReduce



- 50,767,223 nodi
- 2,669,502,959 archi
- 44 GB spazio su disco

Sviluppi futuri

- Matching massimo
- Cammino minimo
- Algoritmi per grafi sparsi
- Lower bounds

In conclusione

- Tecnica innovativa
- Bound limitati
- Nuovo strumento per MapReduce

Grazie per l'attenzione!