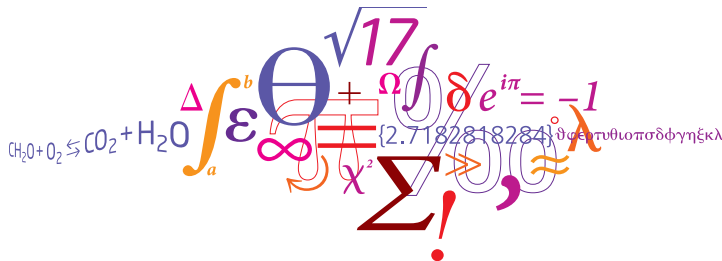


## Algorithm and Programming for Massive Data: Counting Triangles and Closeness Centrality for Graph

Matteo Ghera

Matteo Marulli



# Outline

- Come abbiamo costruito i grafi P ed R
  - Quali sono gli strumenti che abbiamo usato per il progetto
  - R-tree
  - Il grafo delle province italiane
- Conteggio di triangoli
  - Quale problema vogliamo studiare?
  - A che cosa serve contare i triangoli di un grafo?
  - Algoritmi per il conteggio di triangoli
    - Enumerazione delle triple di vertici
    - Enumerazione di coppie di vicini
    - Delega del calcolo ai vertici con grado minore
  - Prestazioni
  - Risultato sul grafico delle province P
- Closeness centrality
  - Definizione
  - Algoritmi per il calcolo della closeness centrality
    - Algoritmo banale
    - Algoritmo RAND
  - Prestazioni
  - Risultato sul grafico delle province P

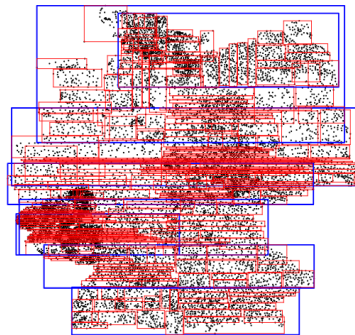



Figure: In case of Coronavirus break this

## Rtree I

Per costruire in modo efficiente il grafo delle province italiane abbiamo usato la struttura dati R-Tree (Guttman, 1984)

- Un R-tree è un albero totalmente bilanciato in altezza simile ad un  $B^+$ -Tree
- I nodi foglia di un R-Tree contengono i record di indice ciascuno dei quali è costituito da una coppia del tipo  $(mbr, tupleId)$
- Per ogni città, la range query  $(lat - d, long - d, lat + d, long + d)$  costa  $\mathcal{O}(\log n)$
- Consentono di eseguire efficientemente le operazioni su database spaziali e vengono usati anche nella computer vision



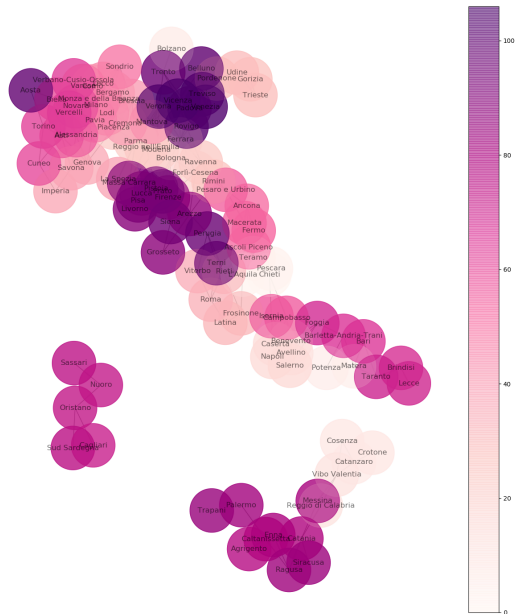
 Guttman A. - *R-trees: A Dynamic Index Structure for Spatial Searching* - Proceedings of the ACM SIGMOD International Conference on Management of Data, 1984

# Come abbiamo costruito i grafi P ed R

## Il grafo delle province italiane I

Il grafo delle province italiane così ottenuto è composto da:

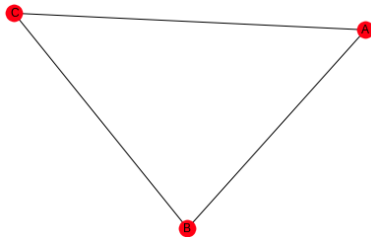
- **numero nodi:** 107
- **numero archi:** 298
- **densità:** 0.0525



Dato un grafo  $G = (V, E)$  indiretto e non pesato.

### Definizione (Triangolo di un grafo)

*Un triangolo in un grafo  $G$  è un insieme di tre vertici che sono mutualmente adiacenti in  $G$*



## Quale problema vogliamo studiare? II

Il primo articolo da noi scelto descrive alcuni algoritmi che possono essere utilizzati per rispondere alle seguenti domande:

- ❶ Quanti triangoli ci sono in  $G$ ?
- ❷ Per ogni vertice  $v \in V$ , quanti triangoli di  $G$  includono il vertice  $v$ ?

**A che cosa serve contare i triangoli del grafo? I**

Dato un grafo  $G = (V, E)$  indiretto e non pesato, si definisce il *coefficiente di clustering* di un vertice  $v$  come la frazione di vicini fra i quali esiste un arco che li collega:

$$C(v) = \frac{|\{u, w \in N(v) | (u, w) \in E\}|}{\binom{\deg(v)}{2}}$$

dove  $\deg(v)$  denota il grado del vertice  $v$  e  $N(v)$  il suo vicinato.



## A che cosa serve contare i triangoli del grafo? II

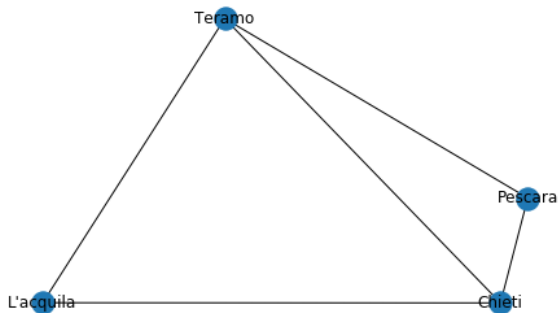


Figure: Il coefficiente di clustering per il nodo "Teramo" è  $2/3$ .

Per calcolare il coefficiente di clustering di un grafo  $C(v)$  dobbiamo determinare il numero di triangoli presenti in  $G$ .

L'articolo scelto propone tre algoritmi:

- l'algoritmo che enumera le triple di vertici
- l'algoritmo che enumera le coppie di vicini
- l'algoritmo che delega il conteggio ai vertici di grado basso

---

**Algorithm 1:** Enumerating over vertex Triples

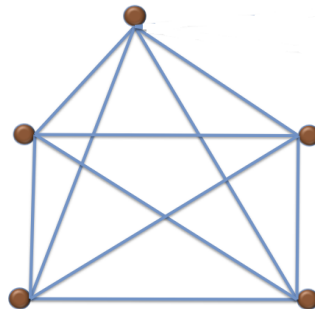
---

**Result:**  $T$ : Number of triangles for each  $v \in V$  $T = \{\}$ **for**  $v \in V$  **do**    **for**  $u \in V \setminus \{v\}$  **do**        **for**  $w \in V \setminus \{u, v\}$  **do**            **if**  $(u, v), (v, w), (u, w) \in E$  **then**                 $T[v] \leftarrow T[v] + 1$             **end**        **end**    **end****end**

---

Analisi dei costi:

- L'algoritmo genera tutte le triple di nodi e questo costa  $\Theta(n^3)$ 
  - Assumendo che la verifica dell'esistenza di un arco di estremi  $(v_1, v_2)$  costi  $O(1)$
  - Il costo finale è  $\Theta(n^3)$  per qualunque grafo
- In quali casi si può fare di meglio?



---

**Algorithm 2:** Enumerating over Neighbor Pairs

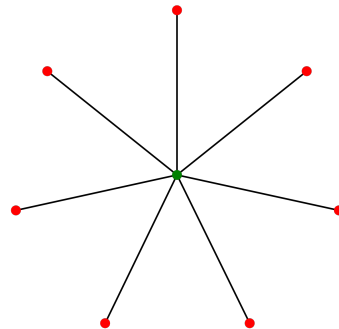
---

**Result:**  $T$ : Number of triangles for each  $v \in V$  $T \leftarrow \{\}$ **for**  $v \in V$  **do**    **for**  $u \in N(v)$  **do**        **for**  $w \in N(v)$  **do**            **if**  $(u, w) \in E$  **then**                 $T[v] \leftarrow T[v] + 1$             **end**        **end**    **end****end**

---

Analisi dei costi:

- Si indichi con  $d(v)$  il grado di un nodo  $v$  allora il costo totale è  $\Theta(\sum_{v \in V} d(v)^2)$
- Il caso pessimo si verifica con con grafi densi ( $n - 1$  archi per ogni nodo) in questo caso il costo è  $\Theta(\sum_{v \in V} d(v)^2) = \Theta(\sum_{v \in V} (n - 1)^2) = \Theta(n(n - 1)^2) = \Theta(n^3)$
- Il caso ottimo si verifica quando ogni vertice di  $G$  ha grado costante e il costo in questo caso è  $O(n)$
- Nei grafi a stella l'algoritmo spende un tempo di  $\Theta(n^2)$  solo sul nodo centrale



## Delega del calcolo ai vertici con grado minore I

Dati due vertici  $u, v \in V$ , la relazione d'ordine  $u \succ v$  indica che il grado di  $u$  è maggiore di  $v$  oppure, qualora i due vertici abbiano lo stesso grado, indica che  $u$  è maggiore di  $v$  rispetto all'ordine lessicografico.

---

### Algorithm 3: Delegating Low-Degree Vertices

---

**Result:**  $T$ : Number of triangles of the network

$T \leftarrow 0$

```
for  $v \in V$  do
  for  $u \in N(v) \wedge u \succ v$  do
    for  $w \in N(v) \wedge w \succ u$  do
      if  $(u, w) \in E$  then
         $T \leftarrow T + 1$ 
      end
    end
  end
end
```

**end**

---

### Teorema

*Assumendo che il tempo necessario ad eseguire un'interrogazione riguardante i collegamenti fra i nodi di un grafo sia costante, l'algoritmo precedente ha un tempo di esecuzione nel caso peggiore dell'ordine di  $O(m^{3/2})$ , dove  $m$  è il numero di archi.*



**Dimostrazione.**

Dividiamo i vertici di  $G$  in due insiemi:

- $\mathcal{H} = \{v \in V : \delta(v) \geq \sqrt{m}\}$
- $\mathcal{S} = \{v \in V : \delta(v) < \sqrt{m}\}$

Il numero di vertici in  $\mathcal{H}$  è al massimo  $2\sqrt{m}$  altrimenti la somma dei gradi dei vertici di  $G$  sarebbe  $\delta(G) = \sum_{v \in V} \delta(v) > 2\sqrt{m} \cdot \sqrt{m} > 2m$  che è impossibile dal momento che  $\delta(G) = \sum_{v \in V} \delta(v) = 2m$ . Il costo per formare le coppie  $(u, w) \in N(v) : \delta(u) > \delta(v) \wedge \delta(w) > \delta(v)$  è  $\mathcal{O}(\delta(v)^2)$ . Quindi il costo totale è dato da:

$$\mathcal{O}\left(\sum_{v \in \mathcal{S}} \delta(v)^2\right) \quad (1)$$

Quanto è grande la (1)?

**Dimostrazione (continua).**

Supponiamo che  $v \in \mathcal{S}$  allora per (1) abbiamo i seguenti vincoli:  $\delta(v) \leq \sqrt{m}$  e  $\sum_{v \in \mathcal{S}} \deg(v) < 2m$ . Applicando lo stesso ragionamento fatto per determinare il numero di vertici in  $\mathcal{H}$  si ottiene:

$$\mathcal{O}\left(\sum_{v \in \mathcal{S}} \delta(v)^2\right) = \mathcal{O}\left(\sum_{v \in \mathcal{S}} (\sqrt{m})^2\right) = \mathcal{O}(2\sqrt{m} \cdot m) = \mathcal{O}(m^{3/2}).$$

Supponiamo adesso che  $v \in \mathcal{H}$ , ricordando che abbiamo al massimo  $2\sqrt{m}$  vertici, si osservi che il lavoro viene svolto dai vertici di  $\mathcal{H}$  che hanno grado minore di  $v$ . Bisogna quindi cercare tra tutte le possibili triple di nodi di  $\mathcal{H}$  con grado minore di  $v$ . Il costo di questa operazione è:

$$(2\sqrt{m})^3 = 8m^{3/2} = \mathcal{O}(m^{3/2}).$$



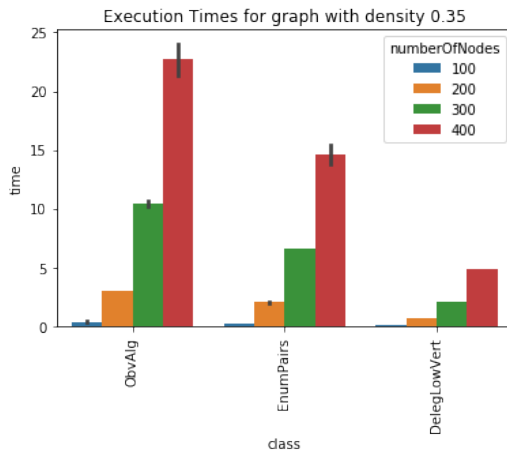


Figure: Tempo di esecuzione degli algoritmi introdotti su grafi con 100, 200, 300, 400 nodi e densità 0.35.

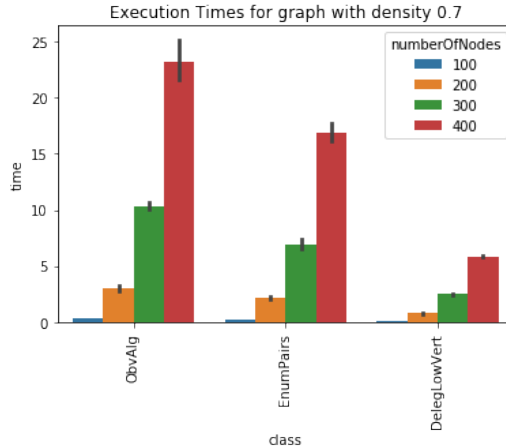
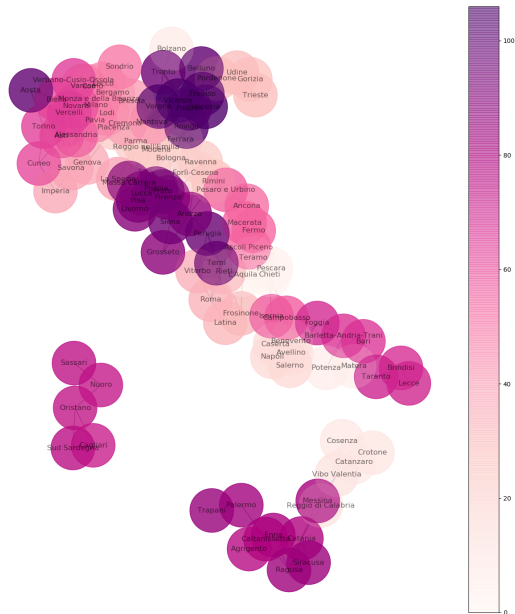


Figure: Tempo di esecuzione degli algoritmi introdotti su grafi con 100, 200, 300, 400 nodi e densità 0.70.

### Risultato sul grafico delle province P I

Il grafo delle province italiane così ottenuto è composto da:

- **numero nodi:** 107
- **numero archi:** 298
- **densità:** 0.0525
- **numero di triangoli:** 352



Provincia	Grado	Numero triangoli	Coefficienti di clustering locale
Milano	13	48	0.6153
Pavia	13	46	0.5897
Monza	13	46	0.5897
Como	12	44	0.6667
Novara	11	36	0.6545
Varese	10	33	0.7333
Lecco	10	32	0.7111
Lodi	10	32	0.7111
Bergamo	10	32	0.7111
Vercelli	10	27	0.6000

Table: Top ten delle province con più triangoli

La closeness centrality di un nodo è un importante indicatore che ci dice quanto un nodo  $v$  consente di raggiungere velocemente gli altri nodi del grafo.

**Definizione (Closeness centrality)**

*In un grafo connesso, la closeness centrality di un nodo  $v$  è definita come*

$$c(v) = \frac{n - 1}{\sum_{u \in V} d(v, u)}$$

## Algoritmi per il calcolo della closeness centrality I

Abbiamo deciso di implementare:

- l'algoritmo che calcola le distanze usando le BFS eseguite partendo da ogni nodo del grafo;
- l'algoritmo RAND di Eppstein e Wang spiegato a lezione.



---

**Algorithm 4:** BFS algorithm

---

**Result:**  $C$ : Closeness centrality for each  $v \in V$

$C \leftarrow \{\}$

**for**  $v \in V$  **do**

$bfsTree \leftarrow BFS(v, G)$

$f_v \leftarrow \text{sum}(bfsTree.getDistance())$

$C[v] \leftarrow \frac{n-1}{f_v}$

**end**

---

Analisi dei costi:

- Per ogni nodo  $v \in V$  eseguiamo una BFS, quindi il costo totale è  $\mathcal{O}(nm)$ ;
- Questo algoritmo non va bene per grafi di grandi dimensioni.

---

**Algorithm 5:** Algorihtm RAND

---

**Result:**  $C$ : Closeness centrality for each  $v \in V$  $C \leftarrow \{\}$  $k \leftarrow \Theta\left(\frac{\log_{10}(n)}{\epsilon^2}\right)$ **for**  $i = 1, \dots, k$  **do**     $v_i \leftarrow$  pick a vertex uniformly random from  $V$     Solve SSSP problem with  $\text{BFS}(v_i, G)$ **end** $f_v \leftarrow 0$ **for**  $v \in V$  **do**    **for**  $i = 1, \dots, k$  **do**         $f_v \leftarrow f_v + d(v, v_i) \cdot \frac{n}{k(n-1)}$     **end**     $C[v] \leftarrow \frac{1}{f_v}$ **end**

---

Analisi dei costi:

- Per ogni  $v_i$  eseguiamo una BFS, i nodi sono  $k$  quindi il costo è  $\mathcal{O}(km)$
- Il costo dei due cicli for successivi è di  $\mathcal{O}(kn)$ . Tuttavia il costo del primo ciclo for è maggiore del costo del secondo.

Correttezza dei risultati:

- sia  $\Delta$  il diametro del grafo e sia  $n$  il numero di nodi del grafo;
- l'errore  $|\hat{c}(v) - c(v)| < \epsilon\Delta, \forall v \in V$  con probabilità  $1 - \frac{1}{n}$ .

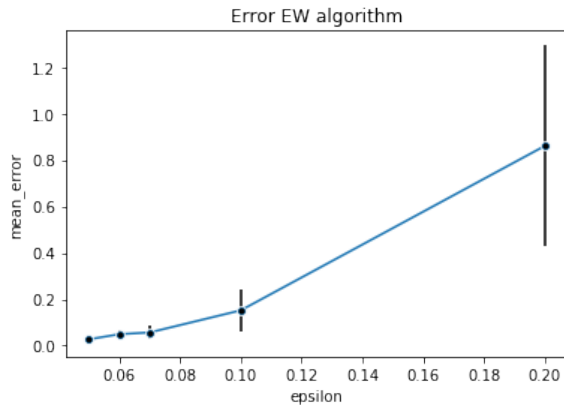


Figure: Andamento dell'errore nei risultati dell'algoritmo RAND al variare di  $\epsilon$

**Risultato sul grafico delle province P I**

Provincia	BFS	EW
Ferrara	0.236 413	0.239 567
Modena	0.231 383	0.234 760
Prato	0.227 154	0.231 661
Rovigo	0.229 551	0.231 661
Arezzo	0.224 227	0.230 142
Forlì-Cesena	0.224 227	0.229 389
Verona	0.227 749	0.229 389
Bologna	0.224 227	0.227 162
Ravenna	0.223 650	0.226 430
Firenze	0.218 045	0.222 130

Table: Top ten delle provincie con la closeness centrality maggiore ottenuta attraverso l'algoritmo EW

# Risultato sul grafico delle province P II

Provincia	BFS	EW
Ferrara	0.236 413	0.239 567
Modena	0.231 383	0.234 760
Rovigo	0.229 551	0.231 661
Verona	0.227 749	0.229 389
Prato	0.227 154	0.231 661
Arezzo	0.224 227	0.230 142
Forlì-Cesena	0.224 227	0.229 389
Bologna	0.224 227	0.227 162
Ravenna	0.223 650	0.226 430
Parma	0.218 045	0.220 041

Table: Top ten delle province con la closeness centrality maggiore ottenuta attraverso l'algoritmo BFS

- L'algoritmo di delegation low-degree vertices è il doppio più veloce dell'algoritmo di enumerating over neighbor pairs sia per grafi densi che per grafi poco densi
- Per grafi di taglia piccola la differenza di velocità non è così grande
- Non spreca tempo su grafi a stella
- In caso di grafi completi l'algoritmo impiega un tempo di  $\mathcal{O}(n^3)$
- L'algoritmo EW si è dimostrato molto veloce nel calcolo approssimato della closeness centrality dei nodi di un grafo
- Per grafi molto grandi dove è possibile scegliere un valore di  $\epsilon$  molto piccolo abbiamo osservato che non solo l'errore medio è basso ma anche la standard deviation. Pertanto più il grafo è grande meglio funziona l'algoritmo nell'approssimazione dei valori di closeness centrality

Bibliografia - Grazie per l'ascolto!!

---

Tim Roughgarden, *CS167: Regading in Algorithms Counting Triangles*, 2014

D. Eppstein and J. Wang, *Fast Approximation of Centrality*. Journal of Graph Algorithms and Applications, 2004