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

Matteo Ghera

Matteo Marulli

Università degli studi di Firenze

Corso di Laurea Magistrale in Informatica Data Science

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?
  - Un approccio matematico: coefficiente di clustering
  - 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
- Closeness centrality
  - Definizione
  - Algoritmi per il calcolo della closeness centrality
    - Algoritmo banale
    - Algoritmo RAND
  - Prestazioni
- Conclusioni

Quali sono gli strumenti che abbiamo usato per il progetto I

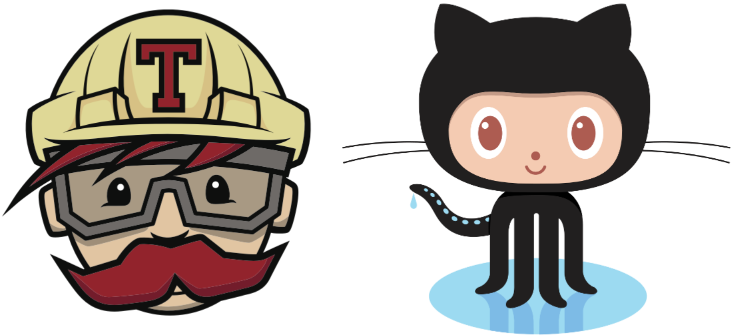
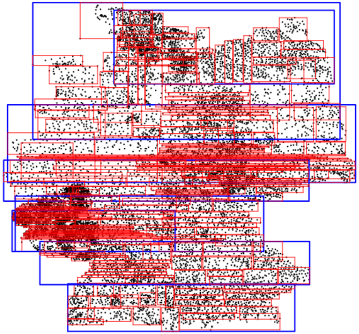


Figure: In case of Coronavirus break this

Rtree I

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

- Un R-tree e un albero totalmente bilanciato in altezza come un B-Tree
- I nodi foglia di un R-Tree contengono i record di indice ciascuno dei quali e costituito da una coppia del tipo (mbr, tupleId)
- La range query costa $\mathcal{O}(\log_M n)$
- Consentono di eseguire efficientemente le operazioni su database spaziali e vengono usati anche nella computer vision

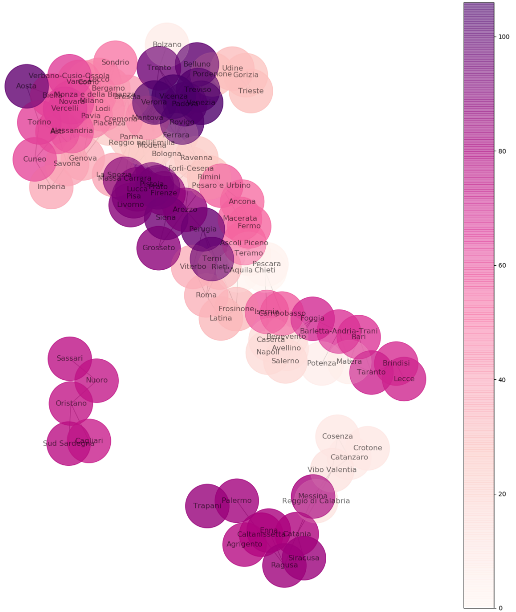


Guttman A. - *R-trees: A Dynamic Index Structure for Spatial Searching* - Proceedings of the ACM SIGMOD International Conference on Management of Data, 1984

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



Quale problema vogliamo studiare? I

Dato un grafo $G = (V, E)$ indiretto e non pesato.
Il primo articolo da noi scelto descrive alcuni algoritmi che possono essere utilizzati per rispondere alle seguenti domande:

1. Quanti triangoli ci sono in G?
2. Per ogni vertice $v \in V$, quanti triangoli di G includono il vertice v?

Ricordando che un *triangolo* in un grafo G è un insieme di tre vertici che sono mutualmente adiacenti in G, si noti che la soluzione del secondo problema ci consente di risolvere immediatamente il primo: si sommano i numeri di triangoli ottenuti per ogni vertice e si divide il risultato per 3.

Un approccio matematico: coefficiente di clustering 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:

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

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

Un approccio matematico: coefficiente di clustering II

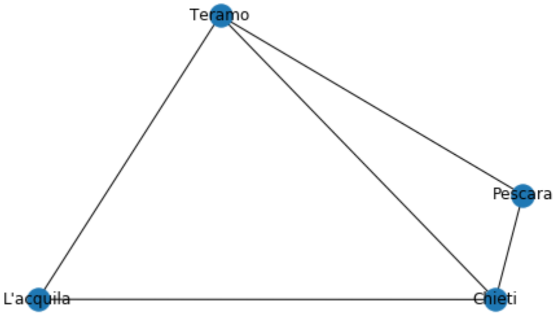
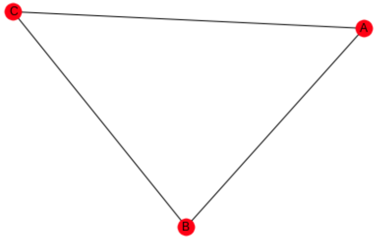


Figure: Il coefficiente di clustering per il nodo 3 è 2/3.

Algoritmi per il conteggio di triangoli I

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:

- Enumerazione di triple di vertici
- Enumerazione di coppie di vicini
- Delegazione per i vertici di grado basso



Enumerazione delle triple di vertici I

Algorithm 1: Enumerating over vertex Triples

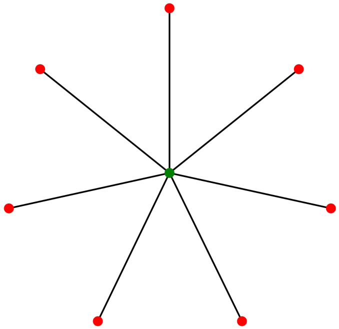
Result: T:Number of triangles for each $v \in V$

```
T = {}
for v ∈ V do
    for u ∈ V \ {v} do
        for w ∈ V \ {u, v} do
            if (u, v), (v, w), (u, w) ∈ E then
                T[v] ← T[v] + 1
            end
        end
    end
end
```

Enumerazione delle triple di vertici II

Analisi dei costi:

- L'algoritmo usa la forza bruta per generare tutte le triple di nodi questo costa $\Theta(n^3)$
- Assumendo che verificare l'esistenza di un arco di estremi (v_1, v_2) costi $O(1)$
- Il costo finale è $\Theta(n^3)$ per qualunque grafo
- In grafi a stella l'algoritmo spende un tempo di $\Theta(n^2)$ solo il nodo centrale



Enumerazione di coppie di vicini I

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

Enumerazione di coppie di vicini II

Analisi dei costi:

- Per ogni nodo $v \in V$ bisogna costruire delle coppie di nodi
- Il grado $d(v)$ di un nodo v corrisponde alla lunghezza del suo vicinato $N(v)$
- Enumerare tutte le coppie di nodi (u, w) da N(v) costa $\Theta(d(v)^2)$
- Il costo totale è $\Theta(\sum_{v \in V} d(v)^2)$
- Il caso pessimo si verifica con con grafi densi $(m = O(n^2))$ 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 un grado costante e il costo in questo caso è $O(n)$

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, u è maggiore di v rispetto all'ordine lessicografico.

Algorithm 3: Delegating Low-Degree Vertices

```
Result: T: Number of triangles of the network
T ← 0
for v ∈ V do
    for u ∈ N(v) ∧ u ≻ v do
        for w ∈ N(v) ∧ w ≻ u do
            if (u, w) ∈ E then
                T ← T + 1
            end
        end
    end
end
```

Delega del calcolo ai vertici con grado minore II

Teorema

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

Delega del calcolo ai vertici con grado minore III

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:

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

Quanto è grande la (1)?

Delega del calcolo ai vertici con grado minore IV

Dimostrazione (continua).

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

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

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:

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

$\square$

Prestazioni I

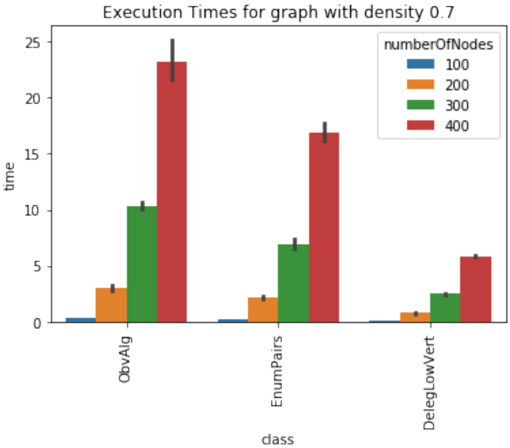


Figure: Tempo di esecuzione degli algoritmi introdotti su grafi con 100, 200, 300, 400 nodi e densità 0.70.

Prestazioni II

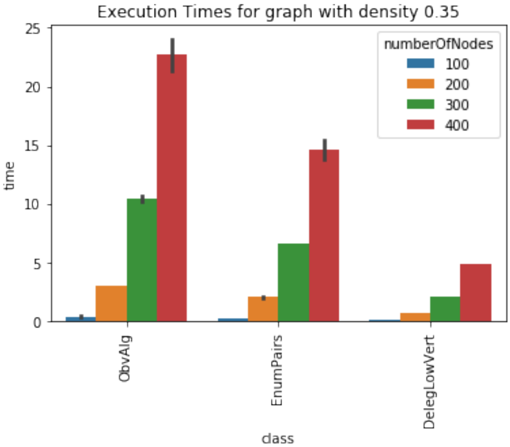


Figure: Tempo di esecuzione degli algoritmi introdotti su grafi con 100, 200, 300, 400 nodi e densità 0.35.

Closeness centrality I

La closeness centrality di un nodo è un importate indicatore che ci dice quanto un nodo v è veloce nel raggiungere 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

Per il calcolo della BFS 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.

Algoritmo banale I

Algorithm 4: BFS algorithm

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

$C \leftarrow \{\}$

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

  $bfsTree \leftarrow BFS(v, G)$

  $f_v \leftarrow 0$

  for $u \in bfsTree$ **do**

    $f_v \leftarrow getDistance(v, u, bfsTree) + f_v$

  end

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

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.

Algoritmo RAND di Eppstein e Wang I

Algorithm 5: Algorihtm RAND

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

$C \leftarrow \{\}$

$k \leftarrow \Theta(\frac{log_{10}(n)}{\epsilon^2})$

for $i = 1, \ldots, k$ **do**

  $v_i \leftarrow$ pick a vertex uniformaly random from V

  Solve SSSP problem with BFS(v_i, G)

end

$f_v \leftarrow 0$

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

  for $i = 1, \ldots, k$ **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

Algoritmo RAND di Eppstein e Wang II

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 Δ 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 $1 - \frac{1}{n}$.

Prestazioni I

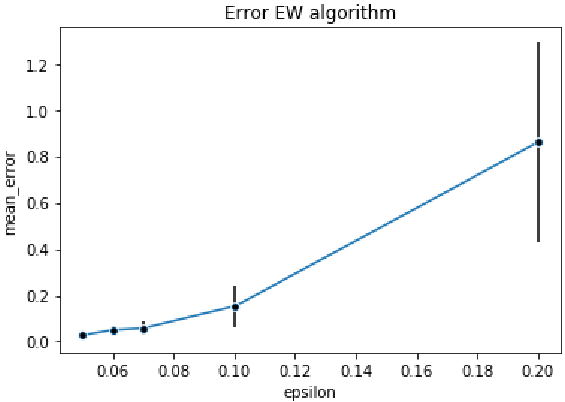


Figure: Andamento dell'errore nei risultati dell'algoritmo RAND al variare di ϵ

Conclusioni I

- L'algoritmo di delegation low-degree vertices è veloce il doppio dell'algoritmo di enumerating over neighbor pairs sia su grafi densi o poco densi
- Per grafi di taglia piccola la differenza di velocità non è cosi 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 ϵ molto piccolo abbiamo osservato che non solo l'errore medio sia basso ma anche la standard deviation, pertanto più il grafo è grande meglio funziona l'algoritmo nell'approssimazione dei valori di closenness 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