Triangle counting

Paolo Boldi
DSI
LAW (Laboratory for Web Algorithmics)
Università degli Studi di Milan

Triangles and clustering

One of the distinguishing features of many complex (undirected) networks is their relatively large *clustering coefficient*:

Triangles and clustering

One of the distinguishing features of many complex (undirected) networks is their relatively large *clustering coefficient*:

- the clustering coefficient of a vertex x is the fraction of its pairs of neighbors that are neighbors of each other

Triangles and clustering

One of the distinguishing features of many complex (undirected) networks is their relatively large *clustering coefficient*:

- the clustering coefficient of a vertex x is the fraction of its pairs of neighbors that are neighbors of each other
- i.e., the fraction of triples (y_1, x, y_2) formed by two edges that form themselves a triangle.

Triangles and clustering

One of the distinguishing features of many complex (undirected) networks is their relatively large *clustering coefficient*:

- the clustering coefficient of a vertex x is the fraction of its pairs of neighbors that are neighbors of each other
- i.e., the fraction of triples (y_1, x, y_2) formed by two edges that form themselves a triangle.
- Social networks exhibit a relatively large clustering coefficient, compared to their diameter.

Local vs. global clustering coefficient

As we said, the *local clustering coefficient* of a vertex x is

$$cc(x) = \frac{|\{\{y,z\}|y,z \in N(x), y \neq z, y \in N(z)\}|s}{\binom{d(x)}{2}}$$

Local vs. global clustering coefficient

As we said, the *local clustering coefficient* of a vertex x is

$$cc(x) = \frac{|\{\{y,z\}|y,z \in N(x), y \neq z, y \in N(z)\}|s}{\binom{d(x)}{2}}$$

A related notion is that of *global clustering coefficient*

$$cc_G = \frac{\sum_x cc(x)}{n},$$

the average clustering coefficient of its vertices.

Local vs. global clustering coefficient

As we said, the *local clustering coefficient* of a vertex x is

$$cc(x) = \frac{|\{\{y,z\}|y,z \in N(x), y \neq z, y \in N(z)\}|s}{\binom{d(x)}{2}}$$

A related notion is that of *global clustering coefficient*

$$cc_G = \frac{\sum_x cc(x)}{n},$$

the average clustering coefficient of its vertices.

- How can one efficiently compute or approximate the local/global clustering coefficient?

Local vs. global clustering coefficient

As we said, the *local clustering coefficient* of a vertex x is

$$cc(x) = \frac{|\{\{y,z\}|y,z \in N(x), y \neq z, y \in N(z)\}|s}{\binom{d(x)}{2}}$$

A related notion is that of *global clustering coefficient*

$$cc_G = \frac{\sum_x cc(x)}{n},$$

the average clustering coefficient of its vertices.

- How can one efficiently compute or approximate the local/global clustering coefficient?
- Here we consider the local case

Triangles of an edge

Define, for every edge yz

$$T(yz) = |N(y) \cap N(z)|.$$

Triangles of an edge

Define, for every edge yz

$$T(yz) = |N(y) \cap N(z)|.$$

This is the number of triangles that the edge yz closes.

Triangles of an edge

Define, for every edge yz

$$T(yz) = |N(y) \cap N(z)|.$$

This is the number of triangles that the edge yz closes.

From this, you can define

$$T(x) = \sum_{y \in N(x)} T(xy),$$

Triangles of an edge

Define, for every edge yz

$$T(yz) = |N(y) \cap N(z)|.$$

This is the number of triangles that the edge yz closes.

From this, you can define

$$T(x) = \sum_{y \in N(x)} T(xy),$$

hence

$$cc(x) = \frac{T(x)}{2\binom{d(x)}{2}}$$

because $T(x)$ counts every triangle twice...

Jaccard coefficient of an edge

The problem thus can be reduced to computing, for every edge yz,

$$T(yz) = |N(y) \cap N(z)|.$$

Jaccard coefficient of an edge

The problem thus can be reduced to computing, for every edge yz,

$$T(yz) = |N(y) \cap N(z)|.$$

Recall the notion of Jaccard coefficient:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccard coefficient of an edge

The problem thus can be reduced to computing, for every edge yz,

$$T(yz) = |N(y) \cap N(z)|.$$

Recall the notion of Jaccard coefficient:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Equivalently:

$$\frac{1}{J(A, B)} = \frac{|A \cup B|}{|A \cap B|} = \frac{|A| + |B| - |A \cap B|}{|A \cap B|} = \frac{|A| + |B|}{|A \cap B|} - 1.$$

Jaccard coefficient of an edge

The problem thus can be reduced to computing, for every edge yz,

$$T(yz) = |N(y) \cap N(z)|.$$

Recall the notion of Jaccard coefficient:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Equivalently:

$$\frac{1}{J(A, B)} = \frac{|A \cup B|}{|A \cap B|} = \frac{|A| + |B| - |A \cap B|}{|A \cap B|} = \frac{|A| + |B|}{|A \cap B|} - 1.$$

Hence

$$|A \cap B| = \frac{|A| + |B|}{1 + \frac{1}{J(A,B)}}.$$

Jaccard coefficient through min-wise permutations

So the problem is further reduced to computing, for every edge yz,

$$J(yz) = J(N(y), N(z)),$$

after which

$$T(yz) = \frac{d(y) + d(z)}{1 + \frac{1}{J(yz)}}.$$

Jaccard coefficient through min-wise permutations

So the problem is further reduced to computing, for every edge yz,

$$J(yz) = J(N(y), N(z)),$$

after which

$$T(yz) = \frac{d(y) + d(z)}{1 + \frac{1}{J(yz)}}.$$

Recall that

Jaccard coefficient through min-wise permutations

So the problem is further reduced to computing, for every edge yz,

$$J(yz) = J(N(y), N(z)),$$

after which

$$T(yz) = \frac{d(y) + d(z)}{1 + \frac{1}{J(yz)}}.$$

Recall that

Theorem

Let $A, B \subseteq \Omega = \{0, 1, \ldots, M-1\}$, and let Π be the set of all $M!$ permutations of Ω. If π is drawn uniformly at random from Π

$$P[\min(\pi(A)) = \min(\pi(B))] = J(A, B).$$

The algorithm (outline)

So, the idea to compute $J(N(y), N(z))$ is:

The algorithm (outline)

So, the idea to compute $J(N(y), N(z))$ is:

- generate a random permutation (i.e., renumbering) π of the nodes

The algorithm (outline)

So, the idea to compute $J(N(y), N(z))$ is:

- generate a random permutation (i.e., renumbering) π of the nodes
- compute $\min \pi(N(y))$ and $\min \pi(N(z))$

The algorithm (outline)

So, the idea to compute $J(N(y), N(z))$ is:

- generate a random permutation (i.e., renumbering) π of the nodes
- compute $\min \pi(N(y))$ and $\min \pi(N(z))$
- if the two values coincide, count $+1$

The algorithm (outline)

So, the idea to compute $J(N(y), N(z))$ is:

- generate a random permutation (i.e., renumbering) π of the nodes
- compute $\min \pi(N(y))$ and $\min \pi(N(z))$
- if the two values coincide, count $+1$

Repeat the above procedue many times, and use the fraction of $+1$'s to estimate $J(N(y), N(z))$.

The algorithm (outline)

Some further notes:

The algorithm (outline)

Some further notes:

- we have a counter per edge $C[yz]$ (to count the number of $+1$'s): we keep them on external memory

The algorithm (outline)

Some further notes:

- we have a counter per edge $C[yz]$ (to count the number of $+1$'s): we keep them on external memory
- to know if $\min \pi(N(y)) = \min \pi(N(z))$ we must have computed the minima before: we need two passes

The algorithm (outline)

Some further notes:

- we have a counter per edge $C[yz]$ (to count the number of $+1$'s): we keep them on external memory
- to know if $\min \pi(N(y)) = \min \pi(N(z))$ we must have computed the minima before: we need two passes
  - first pass: generate the permutation π and compute the minima $\min \pi N(-))$ (kept in central memory)

The algorithm (outline)

Some further notes:

- we have a counter per edge $C[yz]$ (to count the number of $+1$'s): we keep them on external memory
- to know if $\min \pi(N(y)) = \min \pi(N(z))$ we must have computed the minima before: we need two passes
  - first pass: generate the permutation π and compute the minima $\min \pi N(-))$ (kept in central memory)
  - second pass: increment the counters

The algorithm (outline)

Some further notes:

- we have a counter per edge $C[yz]$ (to count the number of $+1$'s): we keep them on external memory
- to know if $\min \pi(N(y)) = \min \pi(N(z))$ we must have computed the minima before: we need two passes
  - first pass: generate the permutation π and compute the minima $\min \pi N(-))$ (kept in central memory)
  - second pass: increment the counters
- we use hashing instead of permutations (equivalent, provided that the probability of collision is negligible).

The algorithm (1)

The algorithm (1)

for K times **do**

The algorithm (1)

for K times **do**

generate a hash function $h : V_G \to \mathbf{N}$

The algorithm (1)

for K times **do**
  generate a hash function $h : V_G \to \mathbf{N}$
  for each $x \in V_G$ **do**

The algorithm (1)

for K times **do**
  generate a hash function $h : V_G \to \mathbf{N}$
  for each $x \in V_G$ **do**
    $M[x] \leftarrow \min_{y \in N(x)} h(y)$
  end for

The algorithm (1)

for K times **do**
  generate a hash function $h : V_G \to \mathbf{N}$
  for each $x \in V_G$ **do**
    $M[x] \leftarrow \min_{y \in N(x)} h(y)$
  end for
  for each $x \in V_G$ **do**
    for each $y \in N(x)$ **do**

The algorithm (1)

for K times **do**
  generate a hash function $h : V_G \to \mathbf{N}$
  for each $x \in V_G$ **do**
    $M[x] \leftarrow \min_{y \in N(x)} h(y)$
  end for
  for each $x \in V_G$ **do**
    for each $y \in N(x)$ **do**
      read $C[xy]$ from disk

The algorithm (1)

for K times **do**
  generate a hash function $h : V_G \to \mathbf{N}$
  for each $x \in V_G$ **do**
    $M[x] \leftarrow \min_{y \in N(x)} h(y)$
  end for
  for each $x \in V_G$ **do**
    for each $y \in N(x)$ **do**
      read $C[xy]$ from disk
      if $M[x] = M[y]$ **then**
        $C[xy] \leftarrow C[xy] + 1$
      end if

The algorithm (1)

for K times **do**
  generate a hash function $h : V_G \to \mathbf{N}$
  for each $x \in V_G$ **do**
    $M[x] \leftarrow \min_{y \in N(x)} h(y)$
  end for
  for each $x \in V_G$ **do**
    for each $y \in N(x)$ **do**
      read $C[xy]$ from disk
      if $M[x] = M[y]$ **then**
        $C[xy] \leftarrow C[xy] + 1$
      end if
      write $C[xy]$ to disk
    end for
  end for
end for

The algorithm (2)

The algorithm (2)

for each $x \in V_G$ **do**
  $T(x) \leftarrow 0$
  for each $y \in N(x)$ **do**

The algorithm (2)

for each $x \in V_G$ **do**
  $T(x) \leftarrow 0$
  for each $y \in N(x)$ **do**
    read $C[xy]$ from disk

The algorithm (2)

for each $x \in V_G$ **do**
  $T(x) \leftarrow 0$
  for each $y \in N(x)$ **do**
    read $C[xy]$ from disk
    $T(xy) \leftarrow (d(x) + d(y))/(1 + K/C[xy])$

The algorithm (2)

for each $x \in V_G$ **do**
  $T(x) \leftarrow 0$
  for each $y \in N(x)$ **do**
    read $C[xy]$ from disk
    $T(xy) \leftarrow (d(x) + d(y))/(1 + K/C[xy])$
    $T(x) \leftarrow T(xy)$
  end for

The algorithm (2)

for each $x \in V_G$ **do**
  $T(x) \leftarrow 0$
  for each $y \in N(x)$ **do**
    read $C[xy]$ from disk
    $T(xy) \leftarrow (d(x) + d(y))/(1 + K/C[xy])$
    $T(x) \leftarrow T(xy)$
  end for
  $cc(x) \leftarrow T(x)/(d(x)^2 - d(x))$
end for