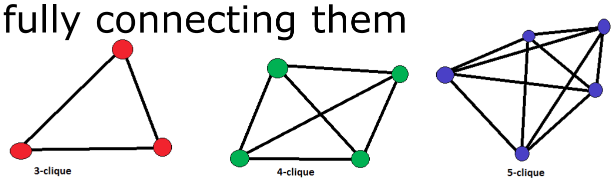
Triangles

Peter M. Kogge

Notation

- **k-Clique**: set of k vertices with k(k-1)/2 edges fully connecting them



- Triangle = 3-clique
- Triangle algorithms:
  - **Find** if *any* triangle exist in a graph
  - **List** *all* triangles in a graph
  - **Count** # of triangles in a graph, but not list
  - **Estimate** # of triangles in a graph

Uses

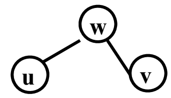
- Finding k-cliques
- Community detection
- Computing clustering coefficients
- Subgraph isomorphism
- Finding minimum circuits

Properties

- For graph G with n vertices, there may be $\theta(n^3)$ or $\theta(m^{3/2})$ triangles
- If vertex v has degree d, at most d(d-1)/2 distinct triangles include it
- Any vertex in a k-clique must be in k-1 triangles with other k-1 vertices
- Each minimum circuit of path length 3 corresponds to a triangle

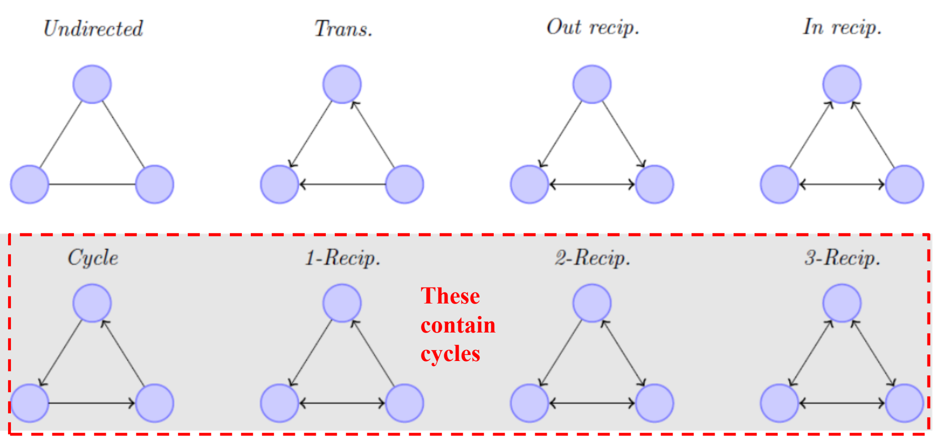
The Importance of Wedges

- **Wedge**:



- If find all wedges, can check each for triangle
- If v has degree d, there are d(d-1) wedges
- High degree vertices have *lots* of wedges
- Common heuristic:
  - "label" all vertices in some order
  - When looking at wedges, check only those where label of u and v are "higher/lower" than w

Taxonomy of Triangles



"Using Triangles to Improve Community Detection in Directed Networks" https://arxiv.org/pdf/1404.5874.pdf

A Trivial Triangle Finder

- For each vertex v
  - Do 3 levels of BFS
  - For each vertex u reached in 3^{rd} level,
    - If u=v then at least one Triangle

Finding Triangles: Matrix Multiply

- Let A = adjacency matrix
  - A[v,u] = 1 if path from u to v
- Consider $Y_2 = A^2$:
  - $Y_2[v,u] = \Sigma A[v,z]*A[z,u]$
  - If $Y_2[v,u] = 1$ and $A[v,u]!=1$ then there is some edge from u to some z, and from z to v
- Consider $Y_3 = A^3$:
  - $Y_3[u,u] = \Sigma A[u,v]*A^2[v,u]$
  - If $Y_3[u,u] = 1$ and $A^2[v,u]!=1$ then there is some edge from u to some z (of length 2), and from z back to u.
  - Total path length = 3 so {u, v,z} forms a triangle
- Time complexity $O(n^{\omega})$, $\omega<2.376$

Finding Triangles: Rooted Trees

- Assume T = a rooted spanning tree in G
  - Every vertex in V is in tree
- Lemma: There is a triangle containing a tree edge iff there is a non-tree edge (u,v) for which (father(u), v) is in E
- Triangle-Finder: repeat until no edges in G
  - Find a rooted spanning tree for each connected component of G
  - If any tree edge is in a triangle (use above) stop
  - If not, delete all edges in tree from G
- $O(M^{3/2})$ time, M = # edges

Listing Algorithm

Algorithm 1 – *forward.* *Lists all the triangles in a graph [25, 26].*

Input: an adjacency array representation of G

1. number the vertices with an injective function $\eta()$ such that $d(u) > d(v)$ implies $\eta(u) < \eta(v)$ for all u and v
2. let A be an array of n arrays initially empty
3. for each vertex v taken in increasing order of $\eta()$:
   - 3a. for each $u \in N(v)$ with $\eta(u) > \eta(v)$:
     - 3aa. for each w in $A[u] \cap A[v]$: output triangle $\{u, v, w\}$
     - 3ab. add v to $A[u]$

order vertices in decreasing order by degree

A[u] contains a set of vertices reachable from u

- $\theta(m^{3/2})$ time, $\theta'(3m+3n)$ space
- Latapy, "Practical algorithms for triangle computation in very large (sparse (power law)) graphs"
- Reduced space ($\theta'(2m+2n)$ by comparing neighbors

Another Listing Algorithm

Algorithm 3 – ***new-listing.*** *Lists all the triangles in a graph.*

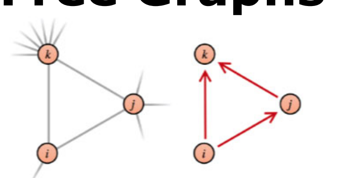
Input: a sorted adjacency array representation of G, and an integer K **K≈√m**

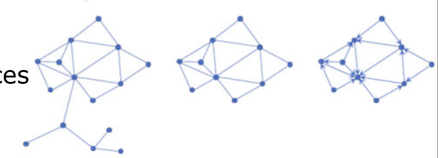
1. for each vertex v in V:
   1a. if $d(v) > K$ then, using the method of Lemma 4:
      1aa. output all triangles $\{v, u, w\}$ such that $d(u) > K$, $d(w) > K$ and $v > u > w$
      1ab. output all triangles $\{v, u, w\}$ such that $d(u) > K$, $d(w) \leq K$ and $v > u$
      1ac. output all triangles $\{v, u, w\}$ such that $d(u) \leq K$, $d(w) > K$ and $v > w$
2. for each edge (v, u) in E:
   2a. if $d(v) \leq K$ and $d(u) \leq K$ then:
      2aa. if $u < v$ then output all triangles containing (u, v) by computing $N(u) \cap N(v)$

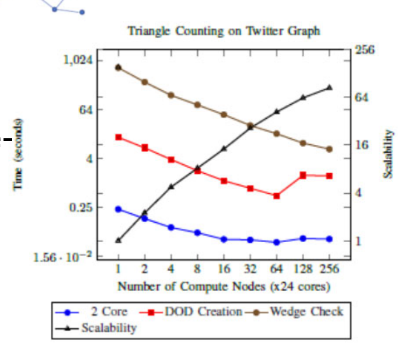
- For power law graphs with exponent α, $\theta(mn^{1/\alpha})$ time
- Latapy, "Practical algorithms for triangle computation in very large (sparse (power law)) graphs"

Counting for Scale-Free Graphs

- **Degree Oriented Directed Graph DOD**: "Augment" graph with new "edges" from low to high degree
  - reduces # of high-degree vertices
  - Reduces # of wedge checks
- Algorithm:
  - Use 2-core to eliminate all vertices not possibly in a triangle
  - Create DOD
  - 1D partition onto nodes
  - Check wedges for each vertex (in parallel)
- Pearse,"Triangle Counting for Scale-Free Graphs at Scale in Distributed Memory", 2017







Parallel Counting

- Partition V into p partitions V_1, V_2, … V_p
- Create subgraphs $V_{i,j,k}$, = V_i U V_j U V_k $i \neq j \neq k$
  - With matching edge subsets: E_{ijk}
- Each triangle must be in at least 1 subgraph
- Load subgraphs on separate nodes
  - Compute # of local triangles
  - Correct for duplicates
- Suri and Vassilvitskii, "Counting Triangles and the Curse of the Last Reducer"