Massive Graph Triangulation

by X. Hu, Y. Tao, and C. Chung, SIGMOD'13

Ilias Giechaskiel

Cambridge University, R212
ig305@cam.ac.uk

February 21, 2014

Conclusions

Takeaway Messages

- Triangle listing important input for graph properties
- I/O becomes bottleneck for massive graphs
  - Obvious approach doesn't work
- MGT algorithm
  - Total order of vertices guarantees unique triangle orientation
  - Near optimal asymptotic I/O + CPU performance
  - Much faster than alternatives in practice

Triangle Listing

Definition

Given a graph $G = (V, E)$, list exactly once all $\Delta_{v_1 v_2 v_3} = \{v_1, v_2, v_3\}$ such that $v_i \in V$ and $(v_i, v_j) \in E$

Motivation

- Triangle = shortest non-trivial cycle and clique
- Various metrics
  - Dense neighborhood discovery
  - Triangular connectivity
  - k-truss
  - Clustering coefficient

In-Memory Triangle Listing [CC12]

The Algorithm

procedure LIST(G)
  $\Delta(G) \leftarrow \emptyset$
  loop $u \in V$
    loop $v \in adj_G(u)$ & $v > u$
      loop $w \in adj_G(u) \cap adj_G(v)$ & $w > v$
        $\Delta(G) \leftarrow \Delta(G) \cup \{\Delta_{uvw}\}$
  return $\Delta(G)$

The Problem

- Random access to $adj_G(v)$ for $v \in adj_G(u)$
- $\mathcal{O}(|E| \cdot scan(d_{max}))$ I/Os in the worst case
  - When it doesn't fit in the memory of size M
  - Recall: $scan(N) = \Theta(N/B)$ where B is the disk block size

Motivation

Previous Approaches

- External Memory Compact Forward (EM-CF)
  - $\mathcal{O}\left(|E| + |E|^{1.5}/B\right)$ I/Os
  - $|E|$ I/O reads
  - Output insensitive
- External Memory Node Iterator (EM-NI)
  - $\mathcal{O}\left(|E|^{1.5}/B \cdot \log_{M/B}\left(|E|/B\right)\right)$ I/Os
  - Almost insensitive to M
  - Output insensitive
- Graph Partition [CC12]
  - $\mathcal{O}\left(|E|^2/(MB) + K/B\right)$ I/Os where K triangles
  - In practice, $M > \sqrt{|E|}$
  - If $M = c|E|$, asymptotically optimal
  - But under a set of assumptions...

Contributions

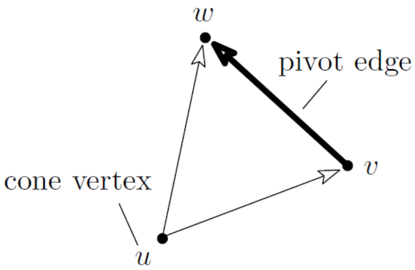
This Approach

- $\mathcal{O}\left(|E|^2/(MB) + K/B\right)$ I/Os in all settings
- $\mathcal{O}\left(|E|\log|E| + |E|^2/M + \alpha|E|\right)$ CPU time
  - α is the arboricity of the graph
- Both optimal up to constants
- Key idea: total order for unique triangle orientation
- Side note: also improves analysis of previous work

Orienting G

Defining G^*

- Define $\prec$ on V by $u \prec v$ iff
  - $d(u) < d(v)$ or $d(u) = d(v)$ and $id(u) < id(v)$
  - Is a total order
- G^* is G with edges oriented by $\prec$
  - Takes $\mathcal{O}(sort(|E|))$ I/Os
  - Recall: $sort(N) = \Theta\left(N/B\log_{M/B} N/B\right)$
- Every triangle $\{u, v, w\}$ has unique orientation $u \prec v \prec w$

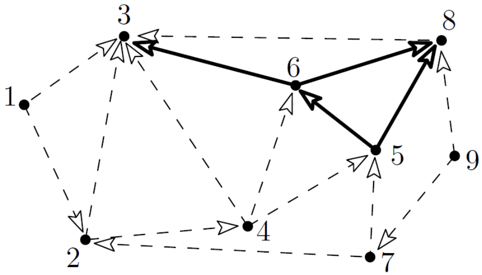


Initial Idea

1. Load next cM edges of G^* into memory (E_{mem})
   - All-or-nothing requirement (small-degree assumption)
2. Find all triangle with pivot edges in E_{mem}

Step 2 (Initial)

procedure LIST(G, E_{mem})
  loop $u \in V$
    $V_{mem}(u) \leftarrow N^{+}(u) \cap V_{mem}$
    Find triangles with u cone in $E_{mem}(u) \cup E_{mem}$



Step 2 (Details)

procedure LIST(G^*, E_{mem})
    Build hash structures
    loop $u \in V$
        $V_{mem}(u) \leftarrow N^+(u) \cap V_{mem}$
        loop $v \in V^+_{mem}(u)$
            loop $w \in V_{mem}(u)$
                if $v \neq w$ & $(v, w) \in E_{mem}$ **then**
                    Output Δ_{uvw}

Analysis

- $\mathcal{O}\left(|E|^2/(MB) + K/B\right)$ I/O
  - $\Theta\left(|E|/M\right)$ iterations
  - $\mathcal{O}\left(|E|/B\right)$ I/Os for scanning
  - $\mathcal{O}\left(K/B\right)$ for listing
- $\mathcal{O}\left(|E|\log|E| + |E|^2/M + \alpha|E|\right)$ CPU
  - $\mathcal{O}\left(|E|\log|E|\right)$ for G^* sorting
  - $\Theta\left(|E|/M\right)$ iterations
  - $\mathcal{O}\left(|N^+(u)| + |N^+(u)| \cdot |V^+_{mem}(u)|\right)$
  - $\Sigma|N^+(u)| = |E|$
  - $\Sigma_{v \in V} d^+(v)^2 = \mathcal{O}(\alpha|E|)$
- Optimality comes from considering the complete graph

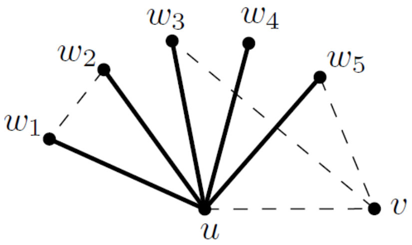
Small-Degree Assumption

- What if $\exists v$ such that $d^+(v) > cM/2$?
  1. Find one
  2. Load a set S of $cM/2$ of its out-edges
  3. Report all triangles involving one of the edges in S
  4. Remove S from the graph
  5. Repeat

The Algorithm

Small-Degree Assumption

- How to implement step 3
  - Create hash table of loaded vertices
  - Scan all $|E|$ edges
  - Also scan $N(v)$ for each $v \neq u$ with $u \in N(v)$
  - Does not change complexity



Evaluation

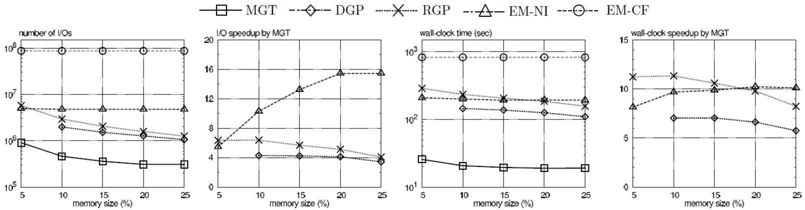
Experimental Setup

- 8GB memory (but memory conscious)
- Graphs unoriented
- Real data
  - 364MB to 7.5GB
  - 4.8 to 165 million vertices
  - 28 to 938 million edges
  - $|E|/|V|$ from 1.2 to 15.1
  - Varied M from 5% to 25% of disk size
- Synthetic data
  - Random, Recursive Matrix, Small World
  - $m = 16n$, n from 16 to 80 million
  - 2.1GB to 10.6GB

Evaluation

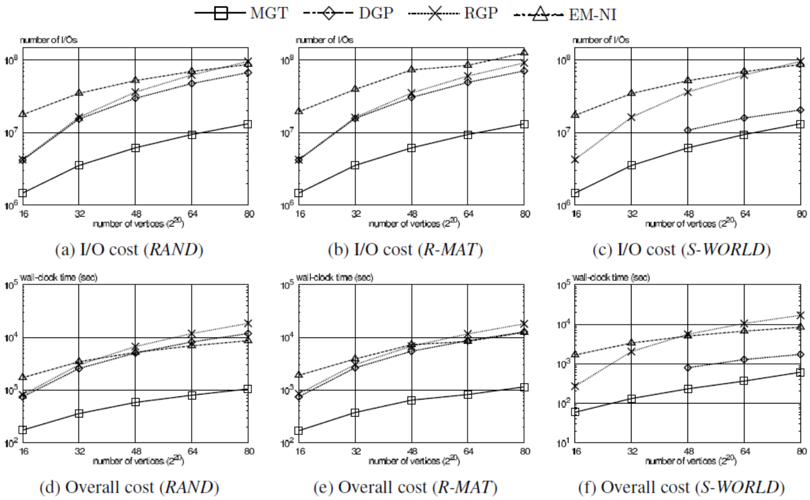
Real Data

- MGT always better for CPU
- MGT almost always better for I/O
- RGP higher hidden constant in complexity!



(a) I/O cost (*LJ*) (b) MGT I/O speedup (*LJ*) (c) Overall cost (*LJ*) (d) MGT overall speedup (*LJ*)

Evaluation


MGT
DGP
RGP
EM-NI
number of I/Os
number of vertices (2^{20})
wall-clock time (sec)
(a) I/O cost (*RAND*)
(b) I/O cost (*R-MAT*)
(c) I/O cost (*S-WORLD*)
(d) Overall cost (*RAND*)
(e) Overall cost (*R-MAT*)
(f) Overall cost (*S-WORLD*)

Criticism

- I/O analysis excludes cost of sorting
- Algorithm does not exploit parallelism
  - Is inherently sequential
  - Not applicable to distributed environment
  - Or across cores
  - RGP ideas applied in this case [PC13]
- Block I/O model for SSDs and parallel environment?
- Behavior for large-degree vertices
- Experiments lacking when M bigger percentage of graph

Conclusions

Key Insights

- Total order of vertices guarantees unique triangle orientation
- Key idea simple, but multiple tricks
- Near optimal asymptotic I/O + CPU performance
- Much faster than alternatives in practice

Key Questions

- Can you parallelize the algorithms non-trivially on a single PC?
- How can you extend the I/O model to different environments?
- How can you minimize data transfers in a distr. environment?
- Your questions?

- Shumo Chu and James Cheng, *Triangle listing in massive networks*, ACM Trans. Knowl. Discov. Data **6** (2012), no. 4, 17:1–17:32.
- Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung, *Massive graph triangulation*, Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (New York, NY, USA), SIGMOD '13, ACM, 2013, pp. 325–336.
- Ha-Myung Park and Chin-Wan Chung, *An efficient mapreduce algorithm for counting triangles in a very large graph*, Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management (New York, NY, USA), CIKM '13, ACM, 2013, pp. 539–548.