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

$\Delta_{v_1v_2v_3} = \{v_1, v_2, v_3\}$ 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
The Algorithm

```plaintext
procedure LIST(G)
    Δ(G) ← ∅
    loop u ∈ V
        loop v ∈ adj_G(u) & v > u
            loop w ∈ adj_G(u) ∩ adj_G(v) & w > v
                Δ(G) ← Δ(G) ∪ {Δuvw}
    return Δ(G)
```

The Problem

- Random access to adj_G(v) for v ∈ adj_G(u)
- \( O(|E| \cdot scan(d_{max})) \) 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)
  - $O(|E| + |E|^{1.5}/B)$ I/Os
  - $|E|$ I/O reads
  - Output insensitive

- External Memory Node Iterator (EM-NI)
  - $O(|E|^{1.5}/B \cdot \log_{M/B}(|E|/B))$ I/Os
  - Almost insensitive to $M$
  - Output insensitive

- Graph Partition [CC12]
  - $O(|E|^2/(MB) + K/B)$ 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

- $O \left( \frac{|E|^2}{MB} + \frac{K}{B} \right)$ I/Os in all settings
- $O \left( |E| \log |E| + \frac{|E|^2}{M} + \alpha |E| \right)$ CPU time
  - $\alpha$ 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 $O(sort(|E|))$ I/Os
  - Recall: $sort(N) = \Theta(N/B\log_{M/B}N/B)$
- Every triangle $\{u, v, w\}$ has unique orientation $u \prec v \prec w$
The Algorithm

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 \textsc{List}(G, E_{\text{mem}}) \\
loop \ u \in\ V \\
\quad V_{\text{mem}}(u) \leftarrow N^{+}(u) \cap V_{\text{mem}} \\
\quad \text{Find triangles with } u \text{ cone in } E_{\text{mem}}(u) \cup E_{\text{mem}}
Step 2 (Details)

**procedure** \textsc{List}(G^*, E_{mem})

Build hash structures

\textbf{loop} \ u \in \ V

\hspace{1cm} V_{mem}(u) \leftarrow N^+(u) \cap V_{mem}

\textbf{loop} \ v \in V^+_{mem}(u)

\hspace{1cm} \textbf{loop} \ w \in V_{mem}(u)

\hspace{2cm} \textbf{if} \ v \neq w \ & \ (v, w) \in E_{mem} \ \textbf{then}

\hspace{3cm} \text{Output } \Delta_{uvw}
The Algorithm

Analysis

- $O\left(\frac{|E|^2}{MB} + \frac{K}{B}\right)$ I/O
  - $\Theta\left(\frac{|E|}{M}\right)$ iterations
  - $O\left(\frac{|E|}{B}\right)$ I/Os for scanning
  - $O\left(\frac{K}{B}\right)$ for listing

- $O\left(|E| \log |E| + \frac{|E|^2}{M} + \alpha|E|\right)$ CPU
  - $O\left(|E| \log |E|\right)$ for $G^*$ sorting
  - $\Theta\left(\frac{|E|}{M}\right)$ iterations
  - $O\left(|N^+(u)| + |N^+(u)| \cdot |V_{mem}(u)|\right)$
  - $\Sigma |N^+(u)| = |E|$
  - $\Sigma_{v \in V} d^+(v)^2 = O(\alpha|E|)$

- Optimality comes from considering the complete graph
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

![Diagram of a graph with vertices and edges]
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
Real Data

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

(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?
