Exact and Parallel Triangle Counting in Dynamic Graphs

Authors: Devavret Makkar, David A. Bader, Oded Green
Slides by: Obada Alkhatib
Triangle Counting Problem

- Given graph $G(V, E)$ with $n$ nodes and $m$ edges, count vertex triplets $(u, v, t)$ s.t. $(u, v), (v, t), (u, t) \in E$.
- 1 permutation of each triplet counted.
Triangle Counting Problem

- Given graph $G(V, E)$ with $n$ nodes and $m$ edges, count vertex triplets $(u, v, t)$ s.t. $(u, v), (v, t), (u, t) \in E$.
- 1 permutation of each triplet counted.

Graph with 2 triangles
Applications

- Clustering coefficient analytic.
- Pattern matching in social networks.
Static Triangle Counting Approaches

- Linear algebra approach involving matrix multiplication - $O(n^\epsilon)$ time, $\epsilon \leq 2.376$
- Adjacency list intersection, complexity $\leq O(m \times d_{\text{max}})$ where $d_{\text{max}}$ is the maximum node degree in $G$. 
Static Triangle Counting Approaches

- Linear algebra approach involving matrix multiplication - $O(n^\epsilon)$ time, $\epsilon \leq 2.376$
- Adjacency list intersection, complexity $\leq O(m \times d_{max})$ where $d_{max}$ is the maximum node degree in G.
Triangle Counting: Dynamic Graphs

- Could count all triangles from scratch after each batch update - very expensive.
- Update triangles of affected vertex due to edge insertion/deletion - still quite expensive.
Triangle Counting: Dynamic Graphs

- Could count all triangles from scratch after each batch update - very expensive.
- Update triangles of affected vertex due to edge insertion/deletion - still quite expensive.
- Idea: update triangle count for affected edge instead - asymptotically less expensive.
STINGER uses blocked linked lists to store edges. This leads to a compromise between low space usage and high data locality.

However, no efficient way for list intersection or sorting.
Used Framework/Data Structure

- STINGER uses blocked linked lists to store edges. This leads to a compromise between low space usage and high data locality.
- However, no efficient way for list intersection or sorting.
- cuSTINGER uses dynamic arrays as adjacency lists. Better locality and suitable for sorting/merging.
Dynamic Graph Updates

- Handle insertions and deletions separately.
- Make temporary update-graph $G' = (V, E')$, where $E'$ is the set of next batch update edges.
- After $G'$ is constructed, sort each adjacency list - which is a dynamic array in cuSTINGER.
- Use fastest possible sorting algorithm (radix sort in the paper, $O(|E'|)$).
Dynamic Graph Updates

- To get output graph, merge corresponding sorted adjacency lists - which cuSTINGER allows efficiently.
- Cost is
  \[ \sum_{(u,v) \in E'} O(d_u^G + d_u^{G'}) \]
Dynamic Graph Updates

- To get output graph, merge corresponding sorted adjacency lists - which cuSTINGER allows efficiently.
- Cost is
  \[ \sum_{(u,v) \in E'} O(d_u^G + d_u^{G'}) \]
- Sometimes, cuSTINGER allows in-place merging.
Dynamic Graph Updates

- To get output graph, merge corresponding sorted adjacency lists - which cuSTINGER allows efficiently.
- Cost is

\[ \sum_{(u, v) \in E'} O(d_u^G + d_u^{G'}) \]

- Sometimes, cuSTINGER allows in-place merging.

![Diagram showing before and after insertion of elements into adjacency lists](image)
Dynamic Graph Updates

- Similar steps for graph deletions, but separate.
- Same overall cost due to use of dynamic arrays:

\[ \sum_{(u,v) \in E'} O(d_u^G + d_u^{G'}) \]
Dynamic Graph Updates

- Similar steps for graph deletions, but separate.
- Same overall cost due to use of dynamic arrays:

\[
\sum_{(u,v) \in E'} O(d_u^G + d_u^{G'})
\]
Dynamic Triangle Counting

- Main challenge is possible new triangles from new and old edges.
- Otherwise would just count triangles in $G'$.
- Three types of new triangles: triangles with 1 new edge ($\Delta^i_1$), triangles with 2 new edges ($\Delta^i_2$), triangles with 3 new edges ($\Delta^i_3$).
Dynamic Triangle Counting

- Main challenge is possible new triangles from new and old edges.
- Otherwise would just count triangles in G’.
- Three types of new triangles: triangles with 1 new edge ($\Delta_1^i$), triangles with 2 new edges ($\Delta_2^i$), triangles with 3 new edges ($\Delta_3^i$).

$$NewTriangles = |\Delta_1^i| + |\Delta_2^i| + |\Delta_3^i|$$
Dynamic Triangle Counting

- Main challenge is possible new triangles from new and old edges.
- Otherwise would just count triangles in $G'$.
- Three types of new triangles: triangles with 1 new edge ($\Delta_1^i$), triangles with 2 new edges ($\Delta_2^i$), triangles with 3 new edges ($\Delta_3^i$).

$$\text{NewTriangles} = |\Delta_1^i| + |\Delta_2^i| + |\Delta_3^i|$$
Dynamic Triangle Counting

- \( s_{e,1} = \text{adj} (u, \widehat{G}_I) \cap \text{adj} (v, \widehat{G}_I) \)
  \[ S_1^i = 2 \cdot |\Delta_1^i| + 4 \cdot |\Delta_2^i| + 6 \cdot |\Delta_3^i| \]

- \( s_{e,2} = \text{adj} (u, \widehat{G}_I) \cap \text{adj} (v, G') \)
  \[ S_2^i = \sum_{e \in E'} |s_{e,2}| = 2 \cdot |\Delta_2^i| + 6 \cdot |\Delta_3^i| \]

- \( S_3^i = 6 \cdot |\Delta_3^i| \)
Dynamic Triangle Counting

- \( s_{e, 1} = \text{adj}(u, \widehat{G}_I) \cap \text{adj}(v, \widehat{G}_I) \)
  \[ S_1^i = 2 \cdot |\Delta_1^i| + 4 \cdot |\Delta_2^i| + 6 \cdot |\Delta_3^i| \]

- \( s_{e, 2} = \text{adj}(u, \widehat{G}_I) \cap \text{adj}(v, G') \)
  \[ S_2^i = \sum_{e \in E'} |s_{e, 2}| = 2 \cdot |\Delta_2^i| + 6 \cdot |\Delta_3^i| \]

- \( S_3^i = 6 \cdot |\Delta_3^i| \)
Dynamic Triangle Counting

\[ \text{NewTriangles} = |\Delta_1^i| + |\Delta_2^i| + |\Delta_3^i| = \frac{1}{2} \left( S_1^i - S_2^i + \frac{S_3^i}{3} \right) \]
Dynamic Triangle Counting

- Deletion simpler - no overcounting, so no inclusion/exclusion.

\[
S^d_1 = 2 \cdot |\Delta^d_1|
\]
\[
S^d_2 = 2 \cdot |\Delta^d_2|
\]
\[
S^d_3 = 2 \cdot |\Delta^d_3|
\]

\[
|\Delta^d_1| + |\Delta^d_2| + |\Delta^d_3| = \frac{1}{2}(S^d_1 + S^d_2 + S^d_3)
\]
Dynamic Triangle Counting

- Complexity analysis:

  \[ O(|E'| \cdot (d_{max}^{G_1} + \hat{d}_{max}^{G_1})) = O(|E'| \cdot \hat{d}_{max}^{G_1}) \]

- Deletion similar.
- Additional optimizations possible, e.g. vertex ordering based on work by Shun & Tangwongsan. Significantly reduces overcounting.
Performance Analysis

- Real-world graphs used.

| Name         | Network Type | $|V|$ | $|E|$ | Ref. | Static (sec.) | Insertion (sec) 100k | Insertion (sec) 1M | Insertion (sec) 10M | Deletion (sec) 100k | Deletion (sec) 1M | Deletion (sec) 10M |
|--------------|--------------|-----|-----|------|----------------|---------------------|-------------------|-------------------|-------------------|------------------|------------------|
| coPapersDBLP | Social       | 540k| 30M | [3]  | 1.032          | 0.053               | 0.452             | -                 | 0.025             | 0.098            | -                |
| in-2004      | Webcrawl     | 1.38M| 27M | [3]  | 18.176         | 0.213               | 2.208             | -                 | 0.117             | 1.805            | -                |
| com-orkut    | Social       | 3M  | 234M| [25] | 90.164         | 0.242               | 1.107             | 10.440            | 0.218             | 0.807            | 8.451            |
| com-LiveJournal | Social     | 4M  | 69M | [25] | 8.975          | 0.168               | 0.765             | -                 | 0.067             | 0.191            | -                |
| cage15       | Matrix       | 5.15M| 94M | [3]  | 1.638          | 0.132               | 0.651             | -                 | 0.043             | 0.091            | -                |
| nlpkkt160    | Matrix       | 8.3M | 221M| [3]  | 1.778          | 0.192               | 0.329             | 7.537             | 0.089             | 0.156            | 0.332            |
| road_central | Road         | 14M | 33M | [3]  | 1.348          | 0.288               | 0.348             | -                 | 0.029             | 0.057            | -                |
| nlpkkt200    | Matrix       | 16.2M| 432M| [3]  | 3.460          | 0.910               | 1.081             | 2.016             | 0.164             | 0.238            | 0.732            |
| road_usa     | Road         | 24M | 58M | [3]  | 2.188          | 0.480               | 0.550             | -                 | 0.046             | 0.074            | -                |
Performance Analysis
Performance Analysis
Conclusion

- Proposed algorithm 100X-819X faster than previous approaches.
- Paper style very straightforward and easy to follow.
- More comparisons to other algorithms might have been more helpful.