Exact and Parallel Triangle Counting in Dynamic Graphs

Authors: Devavret Makkar, David A. Bader, Oded Green

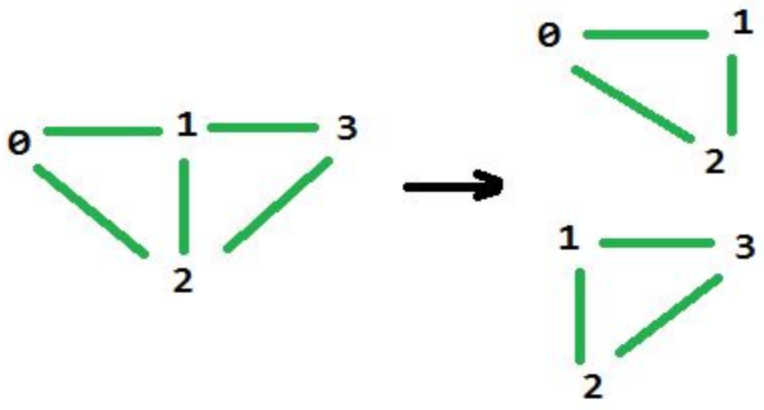
Slides by: Obada Alkhatib

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) ∈ 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) ∈ 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 $\leq O(m \times d_{max})$ where d_{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.

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.

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

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

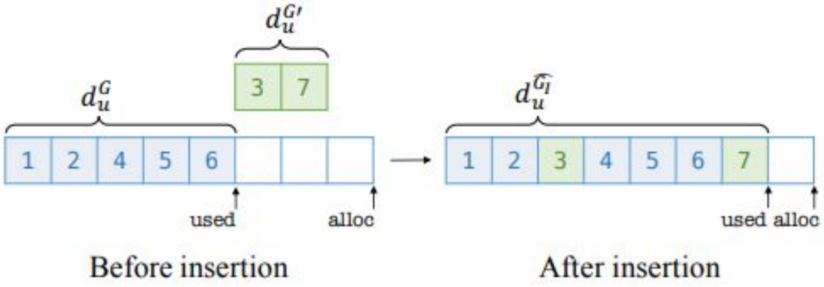
- Sometimes, cuSTINGER allows in-place merging.

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.



(a)

Dynamic Graph Updates

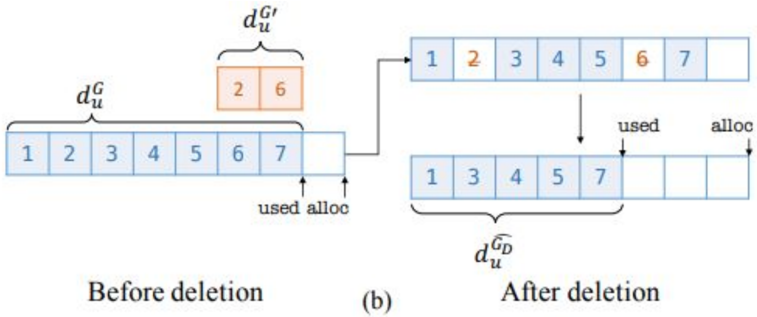
- Similar steps for graph deletions, but separate.
- Same overall cost due to use of dynamic arrays:

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

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 (Δ_1^i), triangles with 2 new edges (Δ_2^i), triangles with 3 new edges (Δ_3^i).

Dynamic Triangle Counting

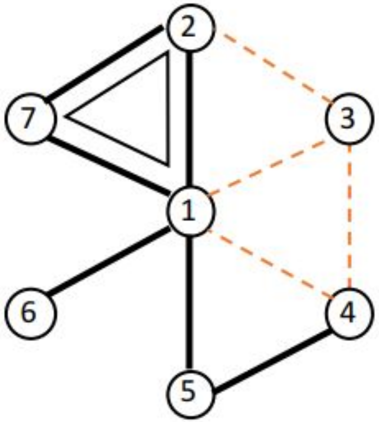
- Main challenge is possible new triangles from new and old edges.
- Otherwise would just count triangles in G’.
- Three types of new triangles: triangles with 1 new edge (Δ_1^i), triangles with 2 new edges (Δ_2^i), triangles with 3 new edges (Δ_3^i).

- $NewTriangles = |\Delta_1^i| + |\Delta_2^i| + |\Delta_3^i|$

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 (Δ_1^i), triangles with 2 new edges (Δ_2^i), triangles with 3 new edges (Δ_3^i).

- $NewTriangles = |\Delta_1^i| + |\Delta_2^i| + |\Delta_3^i|$



Dynamic Triangle Counting

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

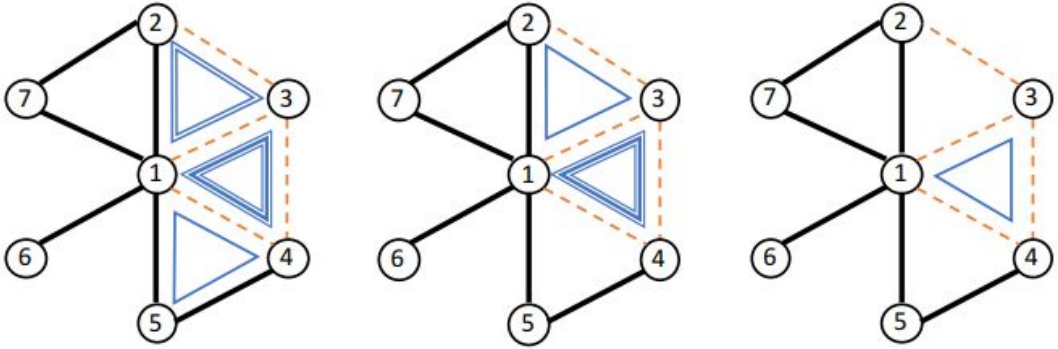
- $s_{e,2} = adj(u, \widehat{G_I}) \cap adj(v, G')$
  $S_2^i = \sum_{e \in E'} |s_{e,2}| = 2 \cdot |\Delta_2^i| + 6 \cdot |\Delta_3^i|$
- $S_3^i = 6 \cdot |\Delta_3^i|$

Dynamic Triangle Counting

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

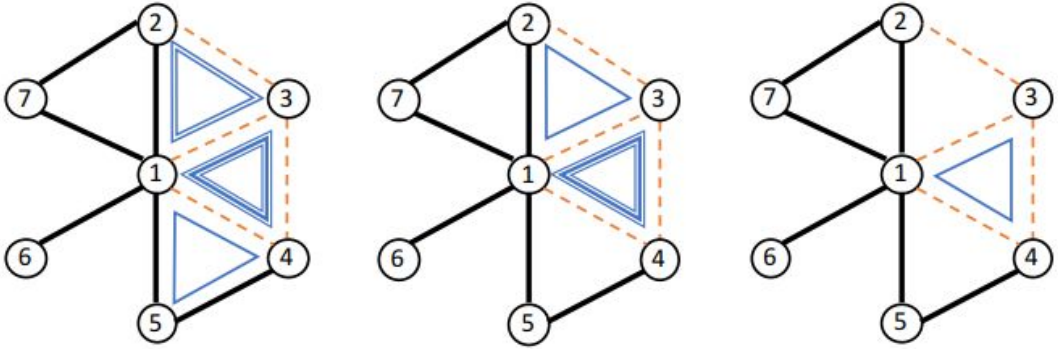
- $s_{e,2} = adj(u, \widehat{G_I}) \cap adj(v, G')$
  $S_2^i = \sum_{e \in E'} |s_{e,2}| = 2 \cdot |\Delta_2^i| + 6 \cdot |\Delta_3^i|$

- $S_3^i = 6 \cdot |\Delta_3^i|$



Dynamic Triangle Counting

$$NewTriangles = |\Delta_1^i| + |\Delta_2^i| + |\Delta_3^i| = \frac{1}{2}\left(S_1^i - S_2^i + \frac{S_3^i}{3}\right)$$



Dynamic Triangle Counting

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

$$S_1^d = 2 \cdot |\Delta_1^d|$$
$$S_2^d = 2 \cdot |\Delta_2^d|$$
$$S_3^d = 2 \cdot |\Delta_3^d|$$

- $$|\Delta_1^d| + |\Delta_2^d| + |\Delta_3^d| = \frac{1}{2}(S_1^d + S_2^d + S_3^d)$$

Dynamic Triangle Counting

- Complexity analysis:

$$O(|E'| \cdot (d_{max}^{\widehat{G_I}} + d_{max}^{\widehat{G_I}})) = O(|E'| \cdot d_{max}^{\widehat{G_I}})$$

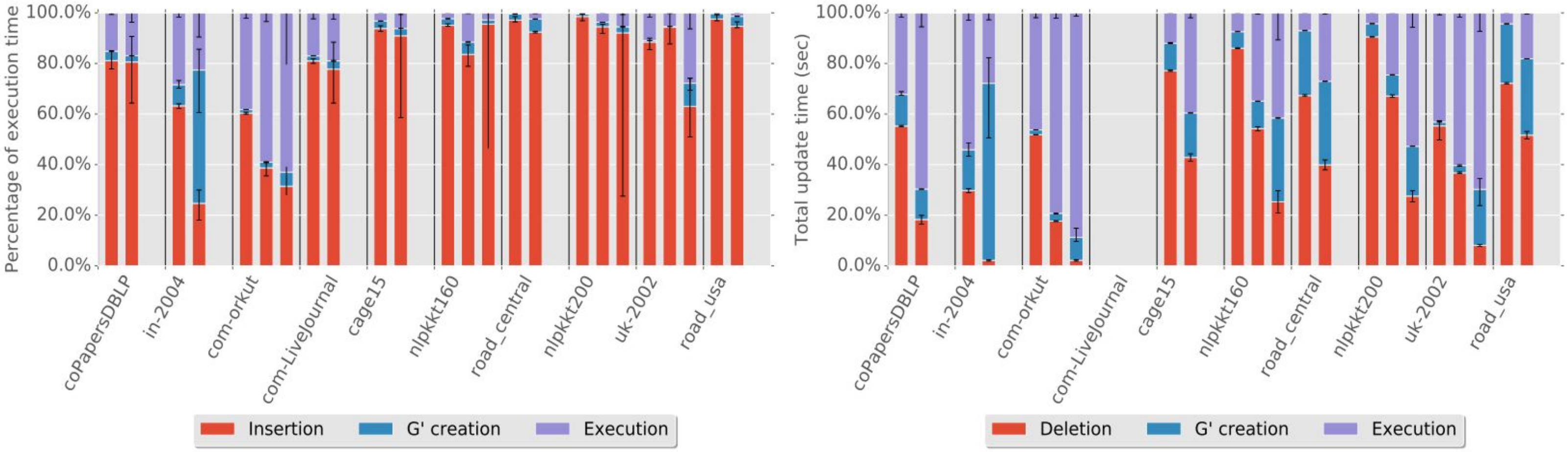
- Deletion similar.
- Additional optimizations possible, e.g. vertex ordering based on work by Shun & Tangwongsan. Significantly reduces overcounting.

Performance Analysis

- Real-world graphs used.

Name	Network Type	$\lvert V\rvert$	$\lvert E\rvert$	Ref.	Static (sec.)	Insertion (sec)			Deletion (sec)		
						100k	1M	10M	100k	1M	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
uk-2002	Webcrawl	18.52M	523M	[3]	522.586	1.653	10.875	12.416	0.629	1.170	5.981
road_usa	Road	24M	58M	[3]	2.188	0.480	0.550	-	0.046	0.074	-

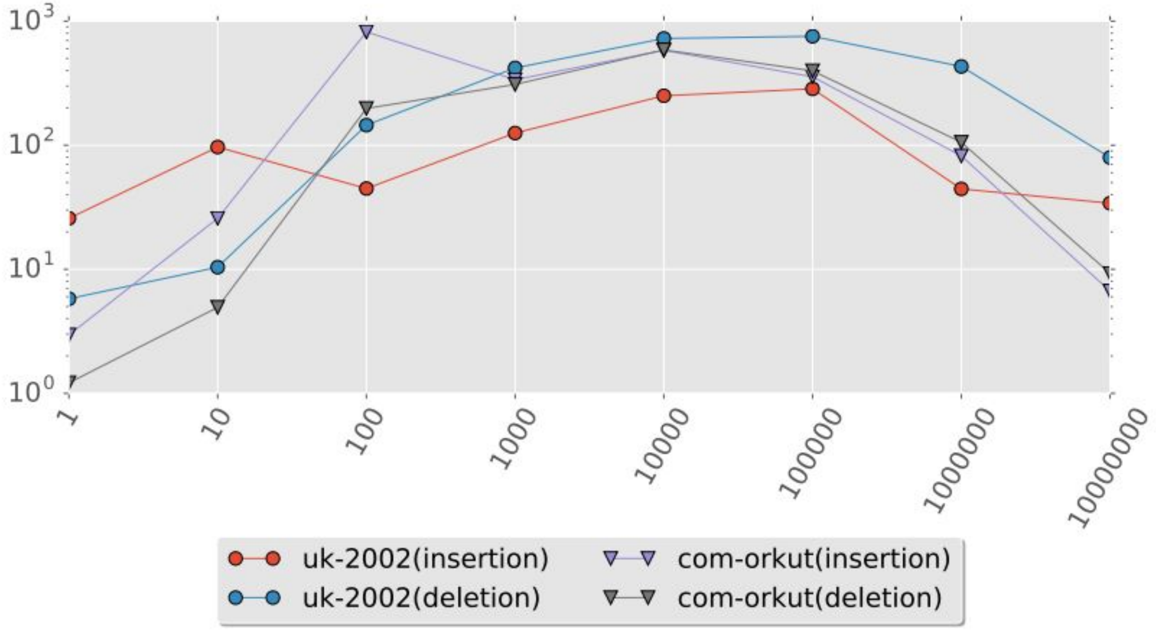
Performance Analysis



(a) Insertions

(b) Deletions

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.