Deriving Streaming Algorithms from Static Definitions

James Fairbanks and David Ediger

Georgia Tech Research Institute

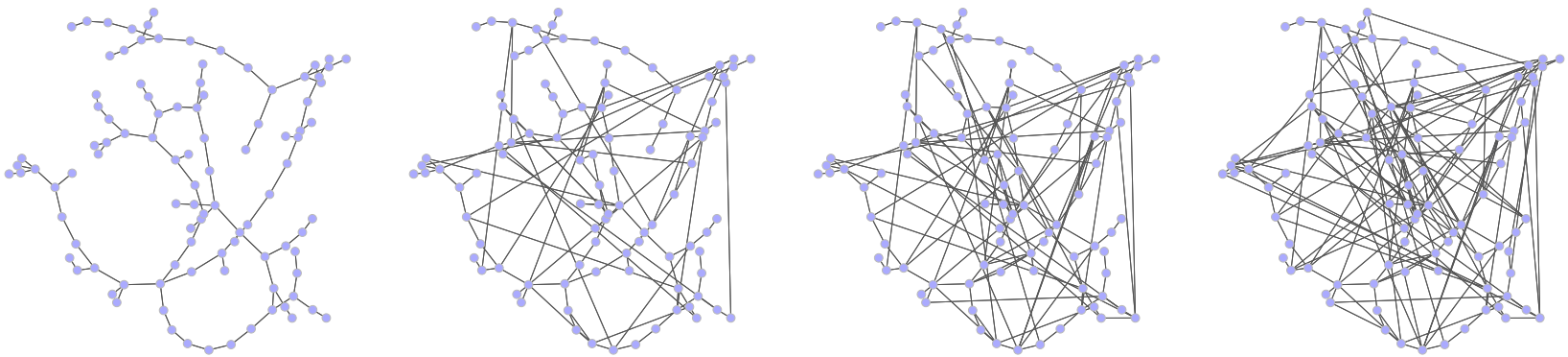
May 28th 2017

Outline

- Incremental Graph Computation
- Triangle Counting
- Performance Results
- Conclusions

Static vs Dynamic

Modern massive networks change quickly.



Examples

- Netflow for cyber activity
- Social Media constantly posting new content

Dynamic Algorithms faster to update than recompute

GABB Should Support Declarative Languages

- Developing high performance graph algorithms is hard
- The GABB system is oriented for static graph algorithms
- We want to derive streaming algorithms from a common definition
- Declarative languages can enable easier high performance graph algorithms

Examples from other domains

- SQL: relational databases [Chaudhuri, 1998]
- AML: algebraic modeling in operations research [Dunning et al., 2015]
- SPARQL: subgraph search in graph databases [Pérez et al., 2006]

Dynamic Graph Computations

- A is the adjacency matrix
- Δ is the adjacency matrix of the update
- $A + \Delta$ is the adjacency matrix of the updated graph G'
- Model of incremental computation:

$$f(A + \Delta) = f(A) + \phi(A, \Delta) \tag{1}$$

- Want to compute $\phi(A, \Delta)$ in closed form

Example: Degree

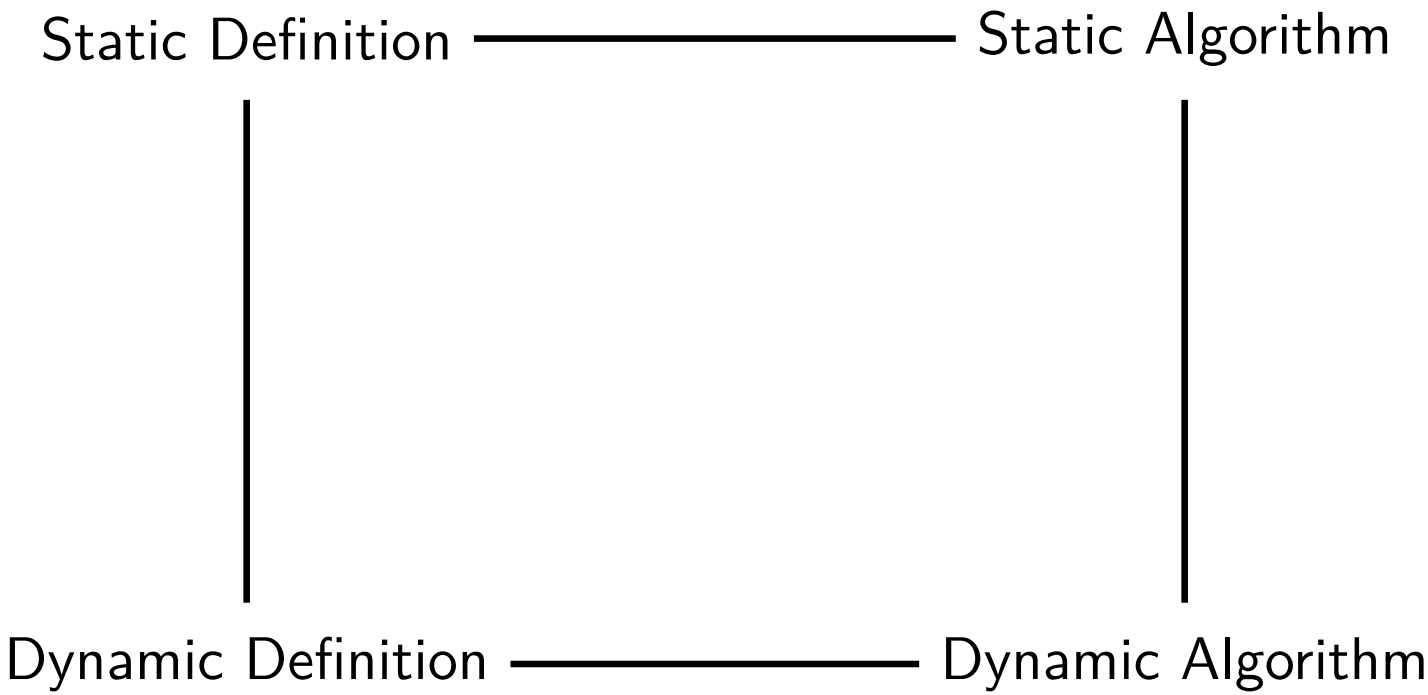
We can compute the streaming degree using linear algebra.

$$deg(G) = A\mathbf{1} \tag{2}$$

$$deg(G') = (A + \Delta)\mathbf{1} = A\mathbf{1} + \Delta\mathbf{1} \tag{3}$$

$$\phi(A, \Delta) = \Delta\mathbf{1} \tag{4}$$

Two Paths to Dynamic Algorithms



Triangle Counting

- Triangles Measure the connectedness of the graph
- Average clustering coefficient is the fraction of wedges in the graph that are closed

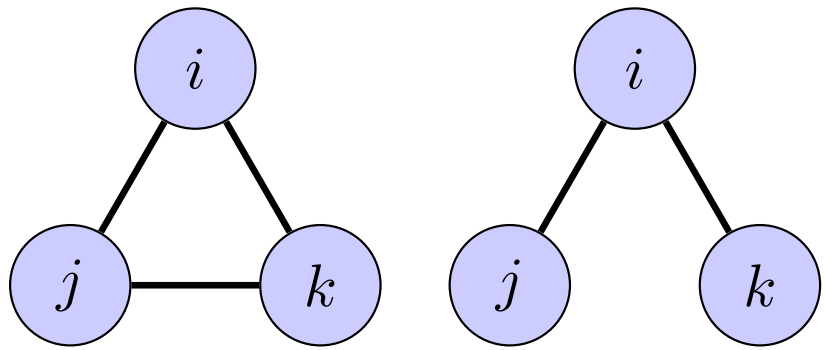


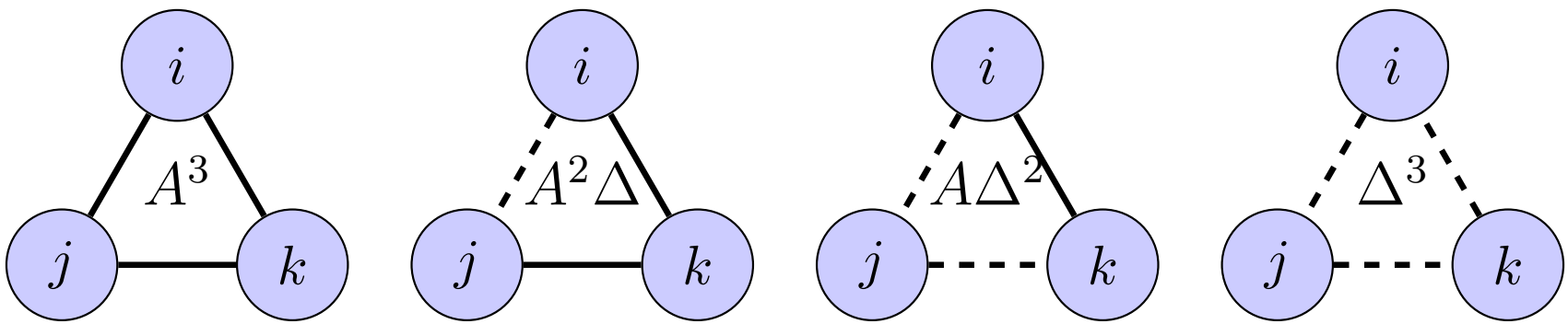
Figure: Triangle (left), Wedge (right)

- Number of triangles is $tr(A^3)$
- Can be computed with "node-iterator" and "edge-iterator" in $O(nd_{max}^2)$ time [Alon et al., 1997, Schank and Wagner, 2005].

Interpretation of Incremental Triangle Computation

$$f(A+\Delta) = f(A) + \phi(A,\Delta)$$

$$tr((A+\Delta)^3) = tr(A^3) + [3tr(A^2\Delta) + 3tr(A\Delta^2) + tr(\Delta^3)]$$



— old
- - new

Model of Incremental Computation is Useful

We can easily:

- Prove correctness unlike manual derivation
- Read off the time static computation time
- Read off the incremental update time
- Compute a break even point

Break Even Point of Incremental Computation

Theorem

Let n, m be number of vertices and edges in the batch of updates δ_{max} is maximum number of updates to a single vertex

The number of triangles in a graph can be updated in

$$O(md_{max} + m\delta_{max}^2 + n\delta_{max}^2)$$

- When $\delta_{max}^2 < d_{max}$, incremental run time is $O(md_{max})$.
- Updating is faster than recomputing when:

$$O(md_{max}) < O(\mid V' \mid d_{max}'^2). \tag{5}$$

- Gives maximum batch size at which it is more efficient to update

Performance Results

$\mid V \mid$	$\mid E \mid$	$tri(A)(ms)$	$tri(A+\Delta)(ms)$	$triupdate(A,\Delta)(ms)$
1000	8000	125.462	114.567	101.206
1000	16000	148.704	164.752	155.811
5000	40000	1291.65	2024.58	742.718
5000	80000	2892.27	3376.18	1645.04
10000	80000	4009.76	5579.61	1518.83
10000	160000	11449.8	12288.0	3704.25
50000	400000	27702.3	46127.7	8452.72

Table 1: Median runtime for computing triangle count statically and dynamically using the trace formulas above. The run time is in milliseconds.

Speedup

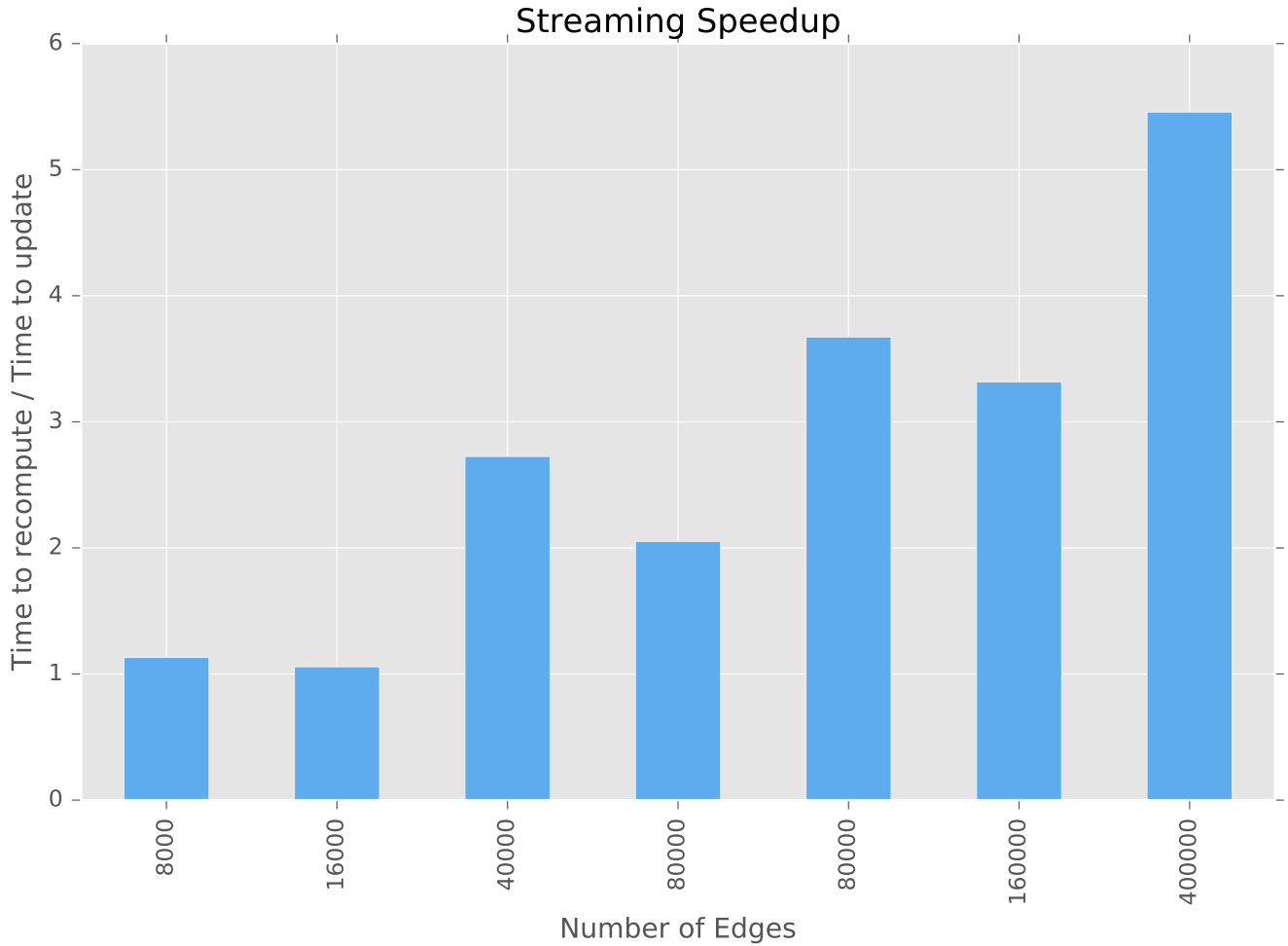


Figure: Performance improvement from streaming triangle updates. Speedup is the time of static computation on the final graph, $tri(A + \Delta)$, divided by time to update, $triupdate(A, \Delta)$

Code Optimizations

```
function triupdate(A, D)
    return  3trace(( A^2 )*D) +
            3trace(A*( D^2 )) +
            trace(D^3) )
end
# reordering to optimize performance
function triupdate2(A, D)
    return  3trace((A*D)*A)
          + 3trace(D*A*D)
          + trace(D^3)
end
```

Figure: `triupdate2` exploits the cyclic permutation of trace to improve efficiency

Optimization Performance Results

$\mid V \mid$	$\mid E \mid$	$triupdate(A, \Delta)(ms)$	$triupdate2(A, \Delta)(ms)$
1000	8000	101.206	105.431
1000	16000	155.811	130.323
5000	40000	742.718	640.394
5000	80000	1645.04	1557.08
10000	80000	1518.83	1365.69
10000	160000	3704.25	3527.96
50000	400000	8452.72	7337.14

Table 2: Median runtime for computing triangle count dynamically using the trace codes above. Note that the run time using `triupdate2` is lower than the run time of `triupdate` indicating that the algebraic transformations described in Figure 3 reduce the run time.

Conclusions

- Expressing graph algorithms in linear algebra leads to dynamic graph algorithms by following perturbations in A
- Algebraic expressions lead to performance optimizations for dynamic graphs
- GABBs are an opportunity to build primitives that enable automatic derivation of streaming graph algorithms

References I

Alon, N., Yuster, R., and Zwick, U. (1997).

Finding and counting given length cycles.

Algorithmica, 17(3):209–223.

Chaudhuri, S. (1998).

An overview of query optimization in relational systems.

In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 34–43, New York, NY, USA. ACM.

Dunning, I., Huchette, J., and Lubin, M. (2015).

JuMP: A modeling language for mathematical optimization.

arXiv:1508.01982 [math.OC].

References II

Pérez, J., Arenas, M., and Gutierrez, C. (2006).

Semantics and Complexity of SPARQL.

In *The Semantic Web - ISWC 2006*, pages 30–43. Springer, Berlin, Heidelberg.

Schank, T. and Wagner, D. (2005).

Finding, counting and listing all triangles in large graphs, an experimental study.

In *Proceedings of the 4th International Conference on Experimental and Efficient Algorithms*, WEA'05, pages 606–609, Berlin, Heidelberg. Springer-Verlag.