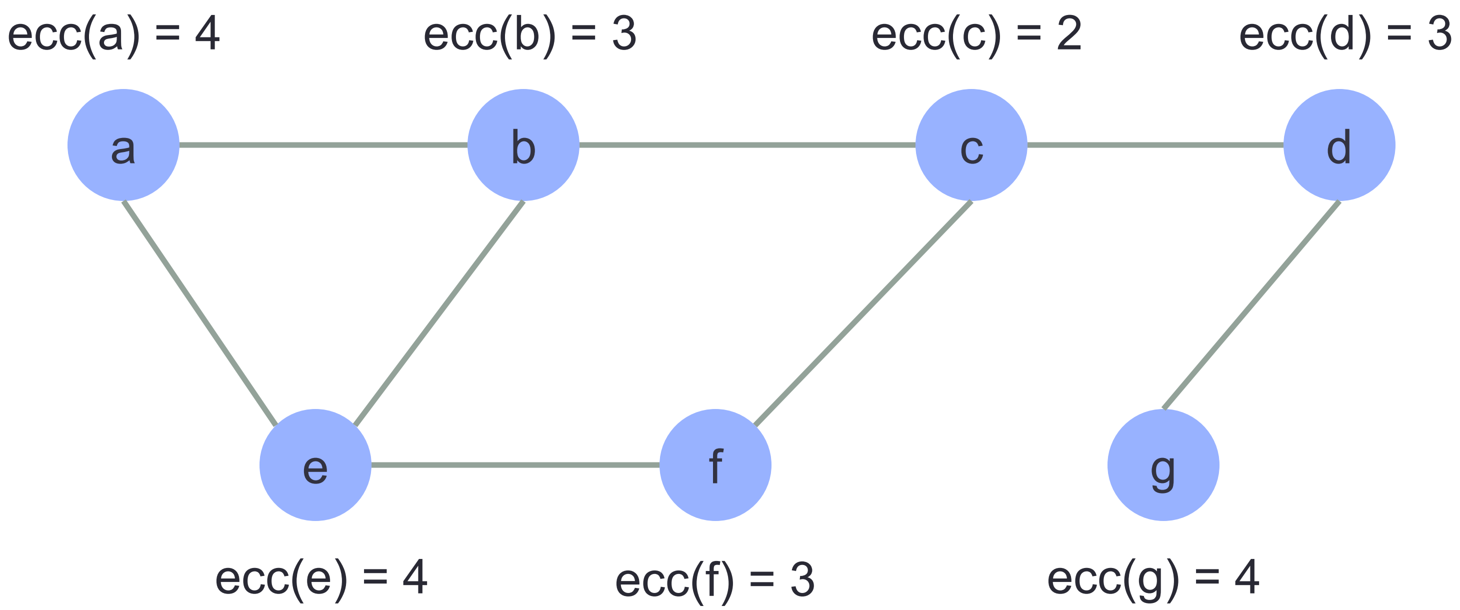
An Evaluation of Parallel Eccentricity Estimation Algorithms on Undirected Real-World Graphs

Julian Shun

Graph Eccentricities

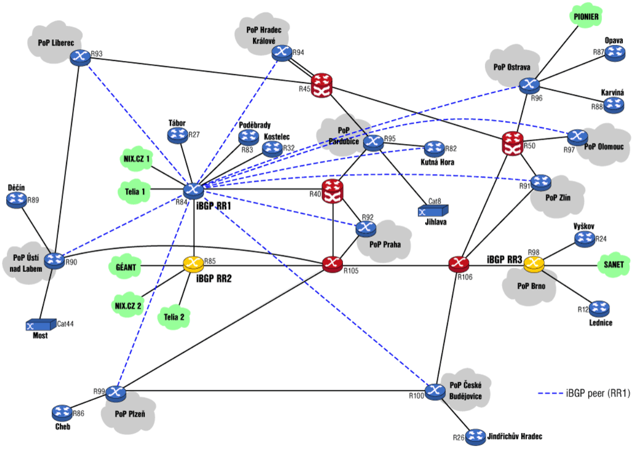
- The **eccentricity** of a vertex v is the distance to furthest reachable vertex from v



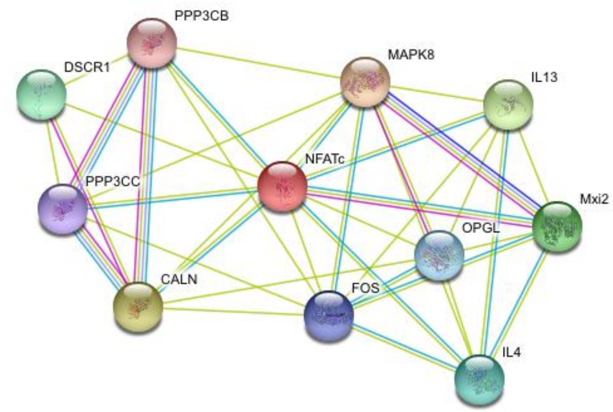
- The **diameter** of a graph is the maximum eccentricity value
- Extends to directed and/or weighted graphs

Applications

Routing networks

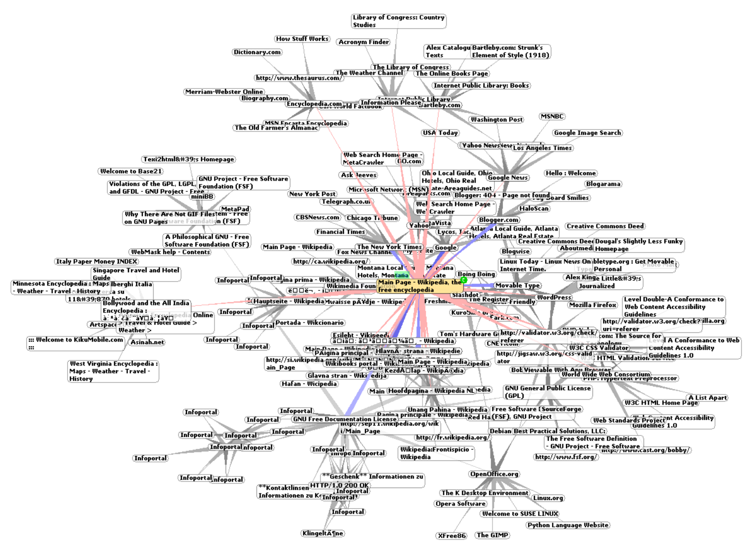


Protein networks



Social networks

Web graphs

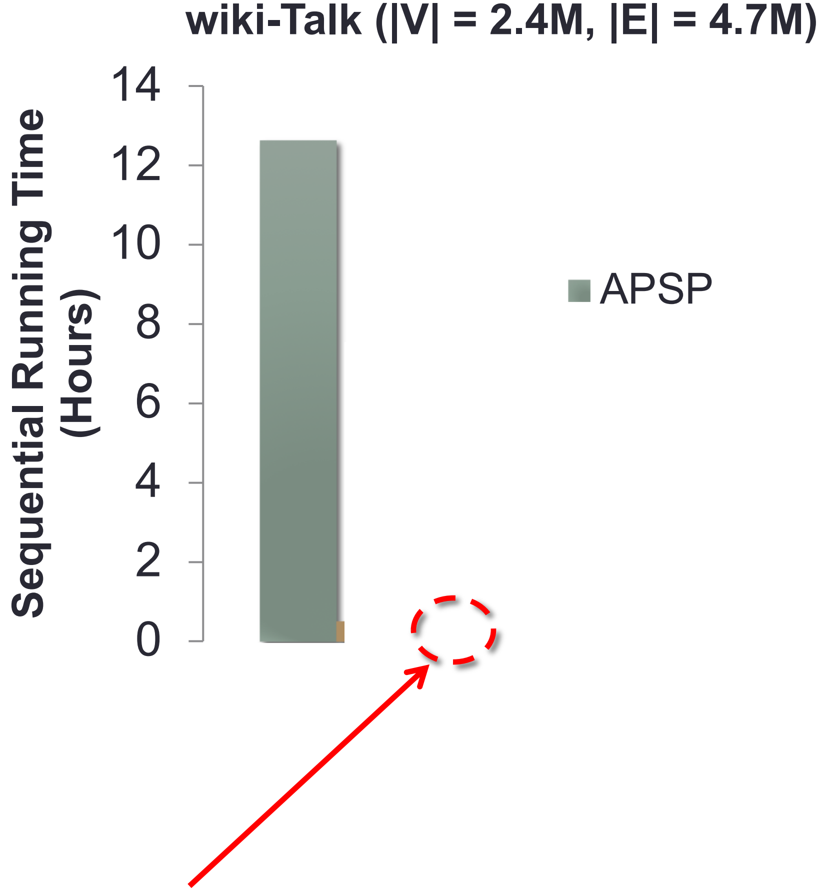


- *Core-periphery structure*

How to compute eccentricities?

Experiments done on one thread of a 40-core (2.4 GHz) Intel Nehalem machine with 256 GB memory

- All pairs shortest paths (APSP)
  - At least quadratic work
  - Would take over 2670 years for Yahoo! Web graph (1.4B vertices, 6.4B edges)
- Takes and Kosters [*Algorithms 2013*] algorithm
  - Quadratic work in the worst case
- Approximation algorithms
  - Orders of magnitude faster
  - Can process Yahoo! Web graph in minutes
- Parallelism



4 seconds, < 0.0001 average relative error

This work

- First comprehensive comparison of parallel implementations of eccentricity algorithms on large (undirected/unweighted) real-world graphs

Algorithm	Guarantee	Complexity
APSP	Exact	O(|E| |V|)
Takes-Kosters [*Algorithms 2013*]	Exact	O(|E| |V|)
Single BFS	2-approx	O(|E|)
Chechik et al. [*SODA 2014*]	1.66-approx	$O(\|E\| (\|V\| \log \|V\|)^{1/2})$
Roditty-Vassilevska Williams [*STOC 2013*]	1.5-approx	$O(\|E\| (\|V\| \log \|V\|)^{1/2})$
ANF/HADI [Palmer et al. *KDD '02*, Kang et al. *TKDD '10*]		O(k |E| D)
HyperANF [Boldi et al. *WWW 2011*]		O(k (log log |V|/log |V|) |E| D)
k-BFS [this work]		O(k |E| min(1,D/log |V|))

|V| = # vertices |E| = # edges k = # probabilistic counters/BFS's D = diameter

- Simple shared-memory implementations in the Ligra graph processing framework [Shun and Blelloch *PPoPP 2013*]
- k-BFS as a parallel primitive for fast, scalable, and accurate eccentricity estimation

Simple 2-approximation

- **Step 1:** Perform breadth-first search (BFS) from arbitrary vertex v to compute eccentricity
- **Step 2**: Assign all vertices ecc(v)
- **Guarantee**: $ecc(w)/2 \le \hat{ecc}(w) \le 2\, ecc(w)$ for all w
- BFS complexity is $O(|E|)$ and easily parallelizable with parallel time proportional to diameter

This algorithm is fast but its estimates are useless!

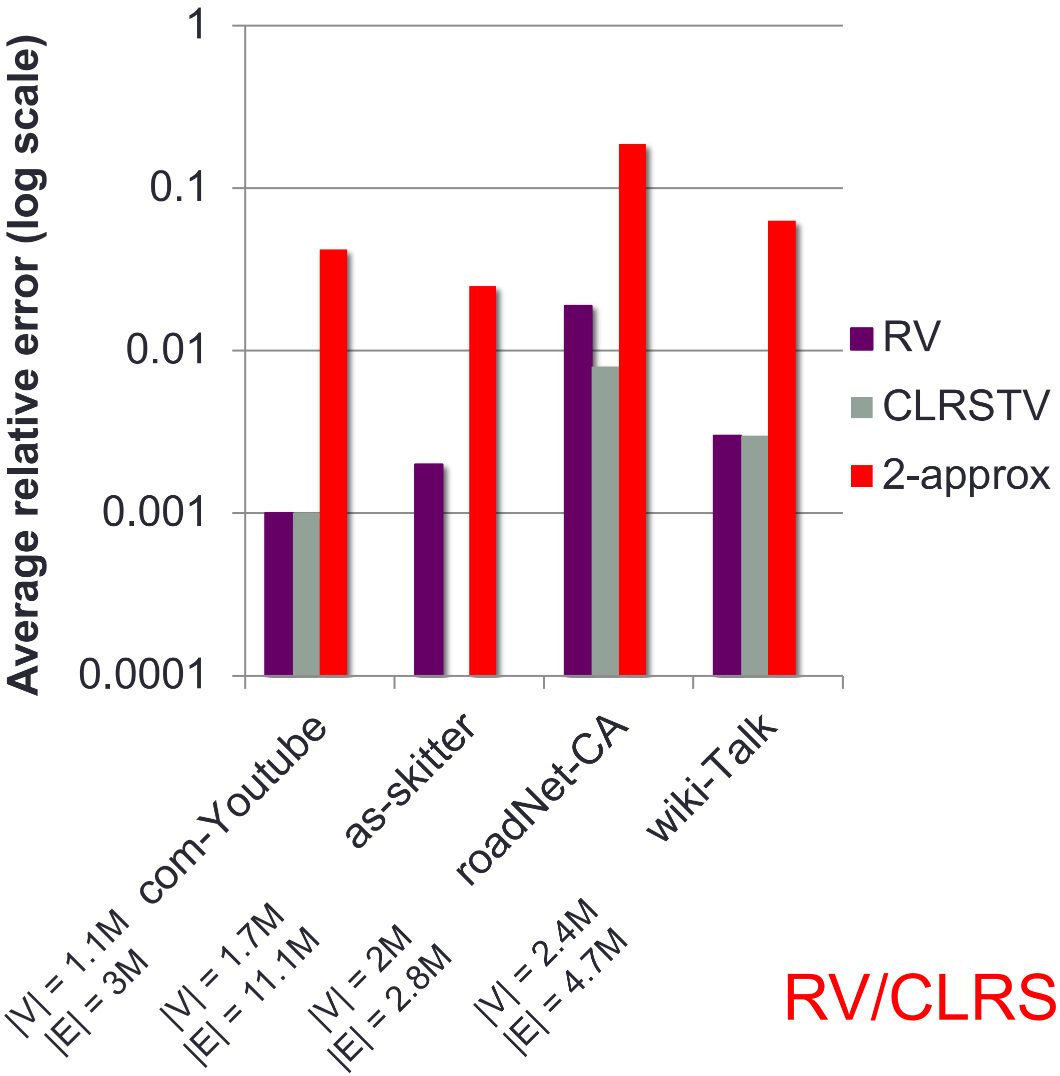
Stronger provable approx algorithms

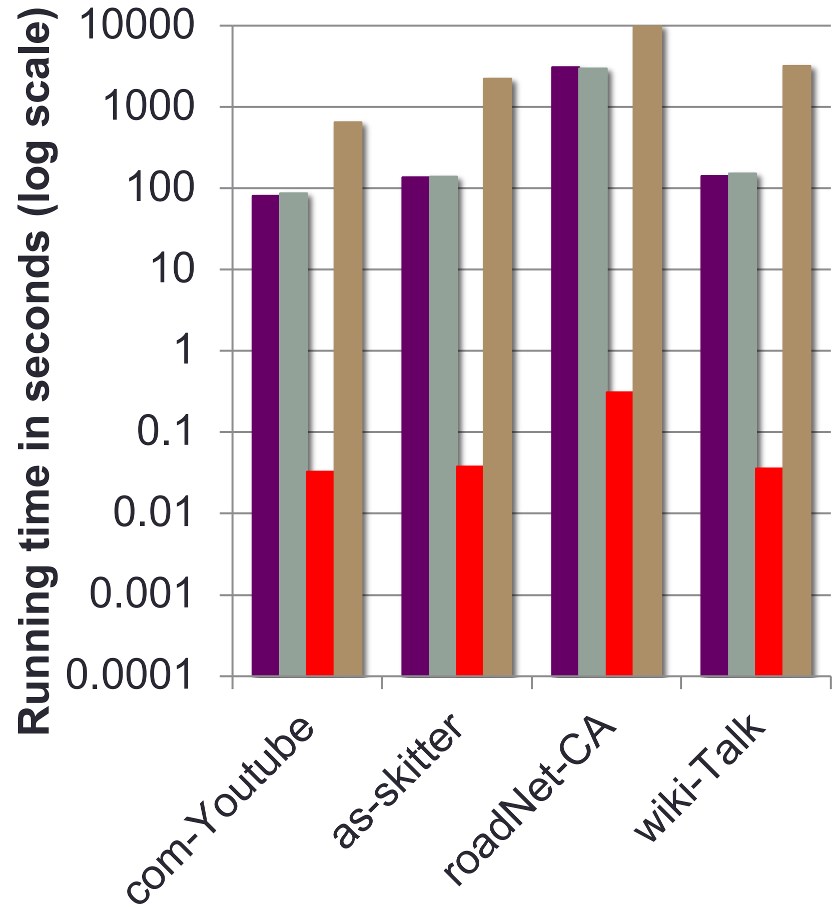
- RV: [Roditty-Vassilevska Williams *STOC 2013*]
  - (2/3) ecc(v) ≤ $\hat{ecc}(v)$ ≤ (3/2) ecc(v)
  - Complexity: $O(|E|\,(|V| \log |V|)^{1/2})$
- CLRSTV: [Chechik, Larkin, Roditty, Schoenebeck, Tarjan, and Vassilevska Williams *SODA 2014*]
  - (3/5) ecc(v) ≤ $\hat{ecc}(v)$ ≤ ecc(v)
  - Complexity: $O(|E|\,(|V| \log |V|)^{1/2})$
- We provide the first empirical evaluation of these algorithms
  - Shared-memory parallel implementations in Ligra

How do they perform in practice?

- Average relative error = $\frac{1}{|V|}\sum_{v \in V}\left|\frac{\hat{ecc}(v) - ecc(v)}{ecc(v)}\right|$

Experiments done on a 40-core Intel Nehalem machine with 256 GB memory

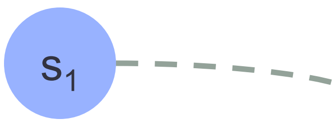


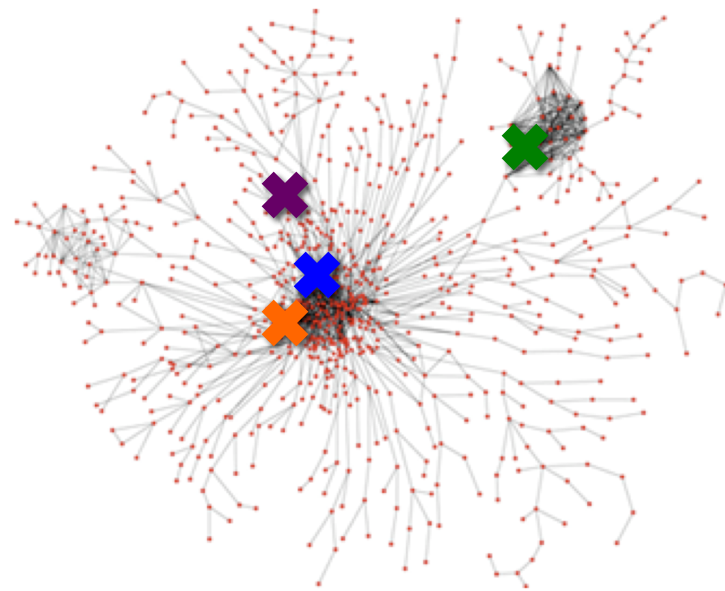


RV/CLRSTV accurate but not scalable

Multiple BFS's

- Run multipl
  vertices anc
  eccentricity





Source: C.A. Hidalgo, B. Klinger, A.-L. Barabási, R. Hausmann. Science **317** (2007)

$$\hat{ecc}(v) = \max(d(v, s_1), d(v, s_2), d(v, s_3), d(v, s_4)) \quad \text{for all } v$$

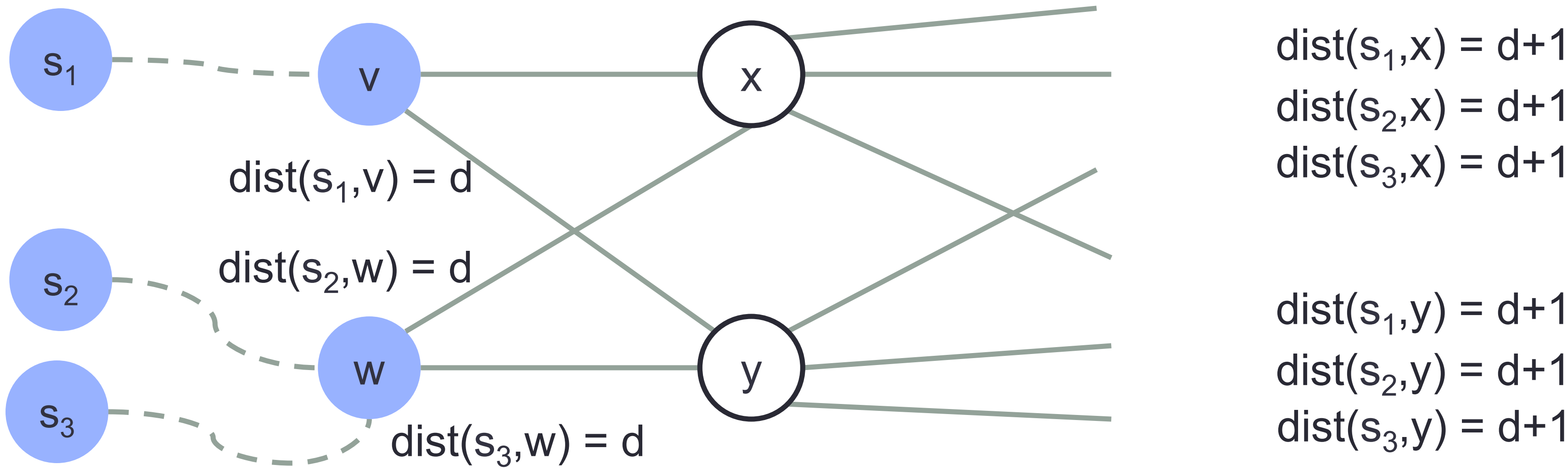
- Does not work that well in practice
- Double-sweep: find furthest vertices from sample of random vertices, then run BFS's from the set of furthest vertices (used before for diameter estimation [Corneil et al. 2001, Magnien et al. 2009])

k-BFS Implementation (second sweep)

- **k** BFS sources
- Each vertex **v** stores estimate $\hat{ecc}$(**v**) = 0
- Run parallel BFS from each source, updating each encountered vertex **v** at distance **d** to max($\hat{ecc}$(**v**), **d**)
- **Observation**: There is shared work among the different BFS's
  - Vertices updated multiple times
  - Vertices placed on frontiers multiple times → edges traversed multiple times

k-BFS Implementation

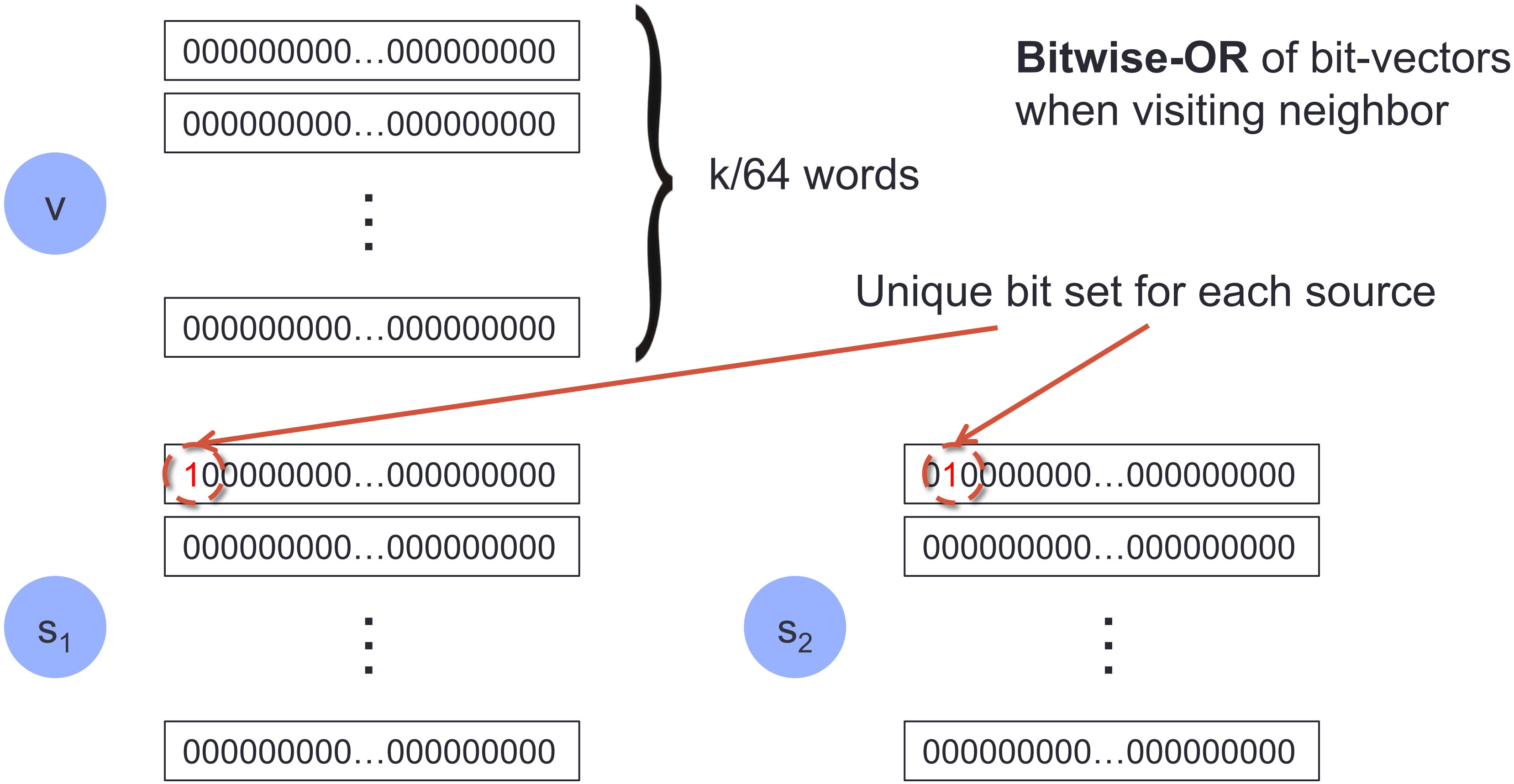
- **Observation**: There is shared work among the different BFS's



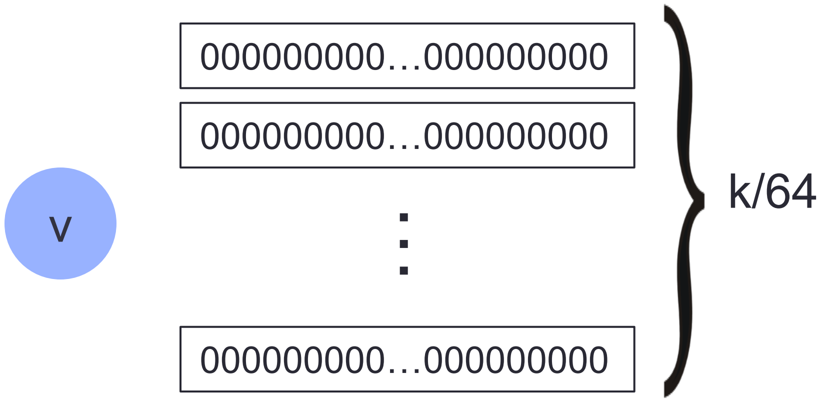
- **Goal**: Reduce redundant computation on vertices
- **Goal**: Reduce the number of times each visited vertex is placed onto the frontier

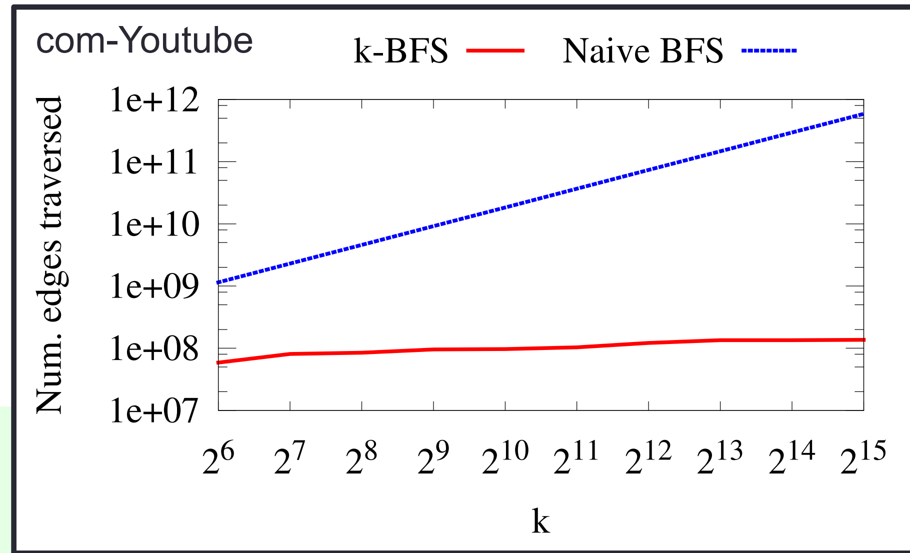
k-BFS Implementation

- Run all k BFS's simultaneously
- Take advantage of bit-level parallelism to store "visited" information



k-BFS Implementation





- Initial **frontier** = $\{s_1, s_2, \ldots, s_k\}$
- d = 0
- While **frontier** not empty: *//Advance all BFS's by 1 level*
  - **nextFrontier** = {}
  - d = d+1
  - For each vertex v in **frontier**:
    - For each neighbor ngh: *//pass "visited" information*
      - Do bitwise-OR of v's words with ngh's words and store in ngh
      - If any of ngh's words changed:
        - eĉc(ngh) = max(eĉc(ngh), d) and place ngh on **nextFrontier** if not there
  - **frontier = nextFrontier**

Parallel k-BFS

- Initial **frontier** = $\{s_1, s_2, \ldots, s_k\}$
- d = 0
- While **frontier** not empty: *//Advance all BFS's by 1 level*
  - **nextFrontier** = {}
  - d = d+1
  - For each vertex v in **frontier**:
    - For each neighbor ngh: *//pass "visited" information*
      - Do bitwise-OR of v's words with ngh's words and store in ngh
      - If any of ngh's words changed:
        - eĉc(ngh) = max(eĉc(ngh), d) and place ngh on **nextFrontier** if not there
  - **frontier = nextFrontier**

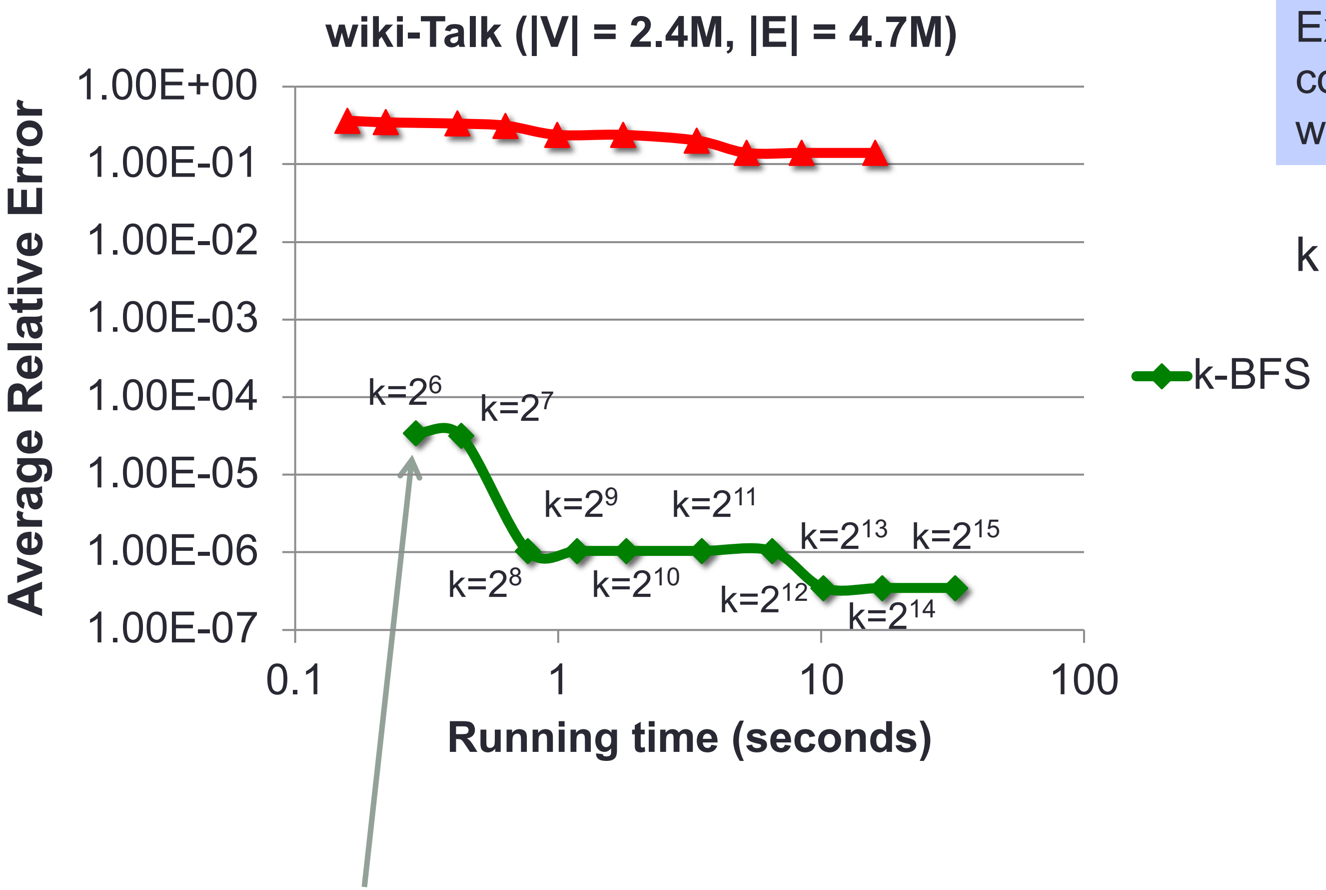
atomic bitwise-OR using compare-and-swap

parallel for-loops

remove duplicates

- Ligra framework that we use takes care of most details
  - User only specifies function to apply on each edge traversed
  - Performs "direction-optimizing" BFS [Beamer '12] automatically

k-BFS Performance (varying k)



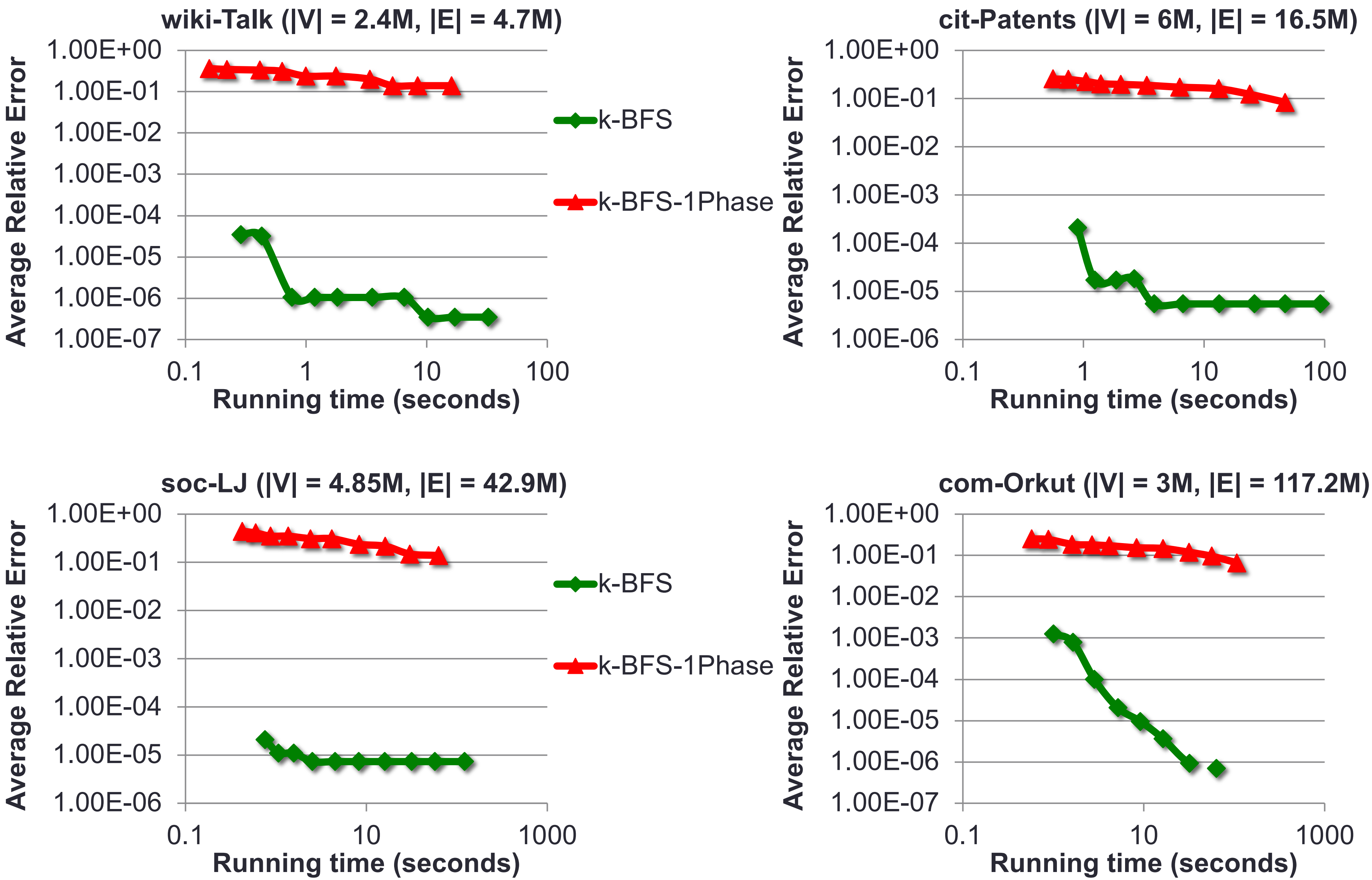
Experiments done on a 40-core Intel Nehalem machine with 256 GB memory

$k = \{2^6, 2^7, 2^8, \ldots, 2^{15}\}$

k-BFS (k = 64)	0.288 sec	$< 10^{-4}$

k-BFS Performance

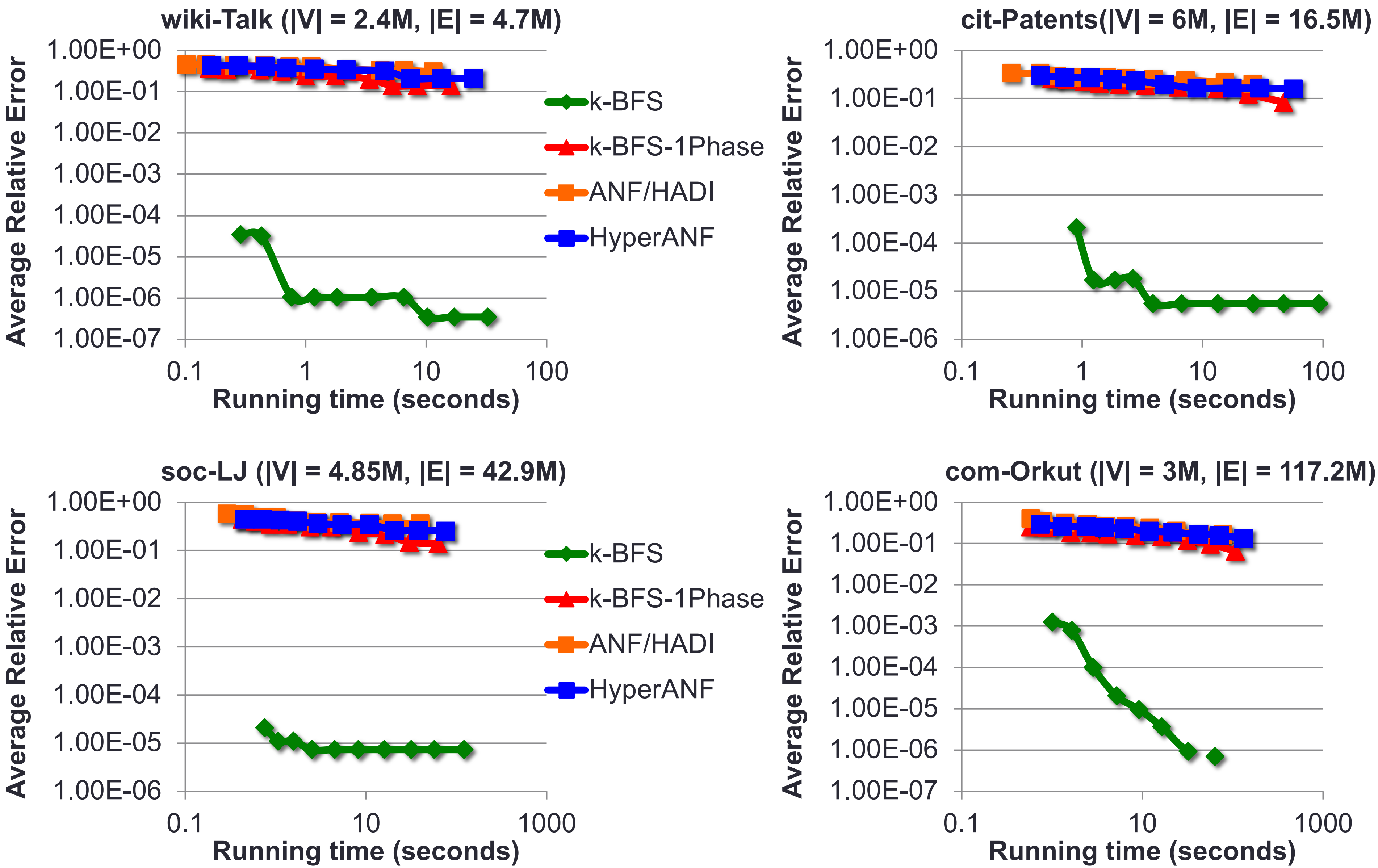
Experiments done on a 40-core Intel Nehalem machine with 256 GB memory



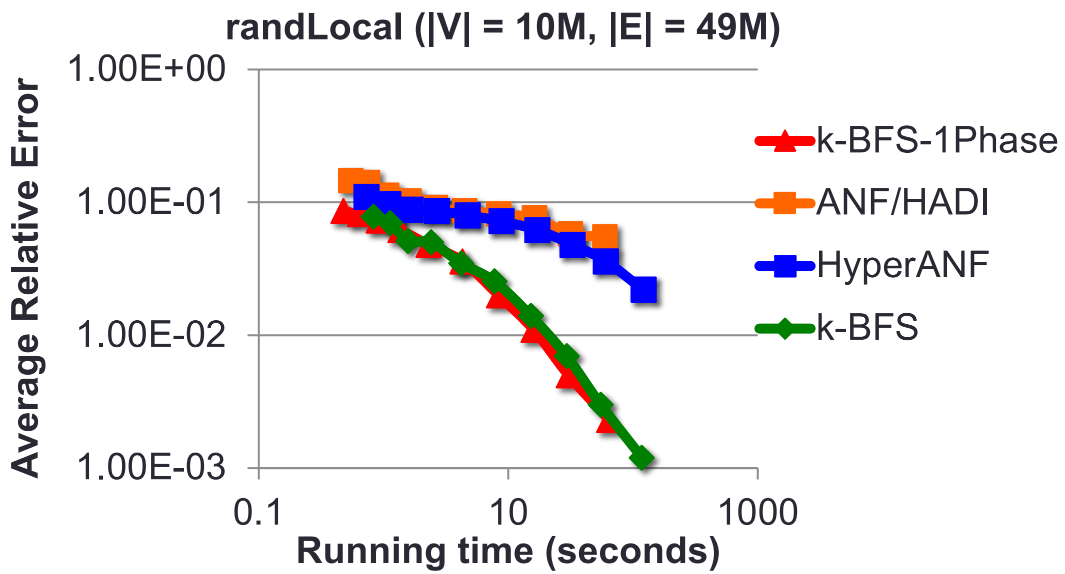
Probabilistic Counter-based Algorithms

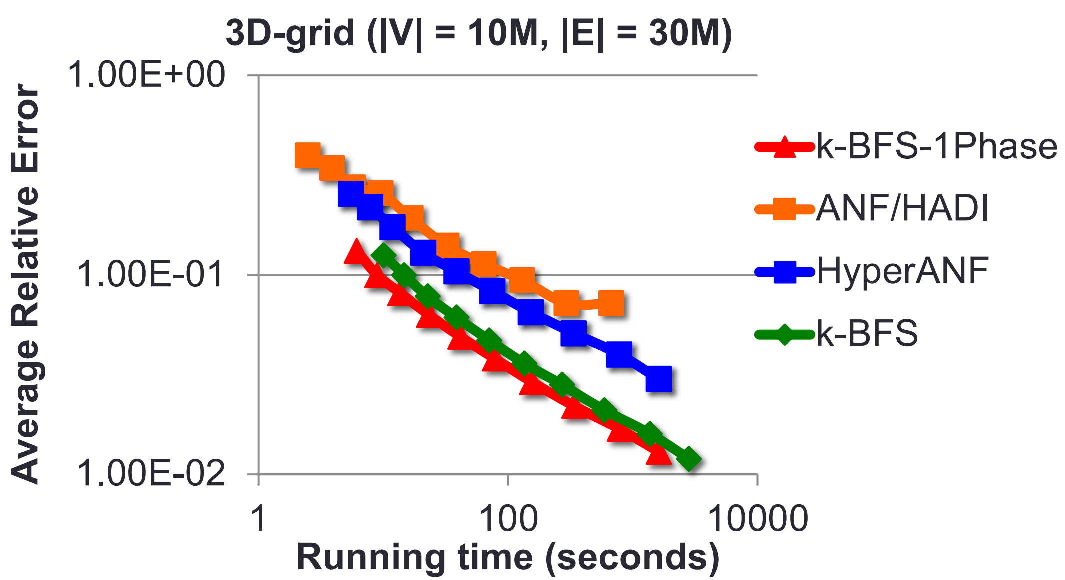
- Assign k probabilistic counters per vertex
- Bitwise-OR counters with all of neighbors' counters until all counters stabilize
- Eccentricity estimate for a vertex is the last round in which any of its counters changed
- ANF/HADI algorithm [Palmer et al. '03, Kang et al. '10] used Flajolet-Martin counters
- HyperANF algorithm [Boldi et al. '11] use more space-efficient HyperLogLog counters [Flajolet et al. '08]
- Shared-memory implementations of variants of ANF/HADI and HyperANF in Ligra

k-BFS outperforms ANF/HADI and HyperANF



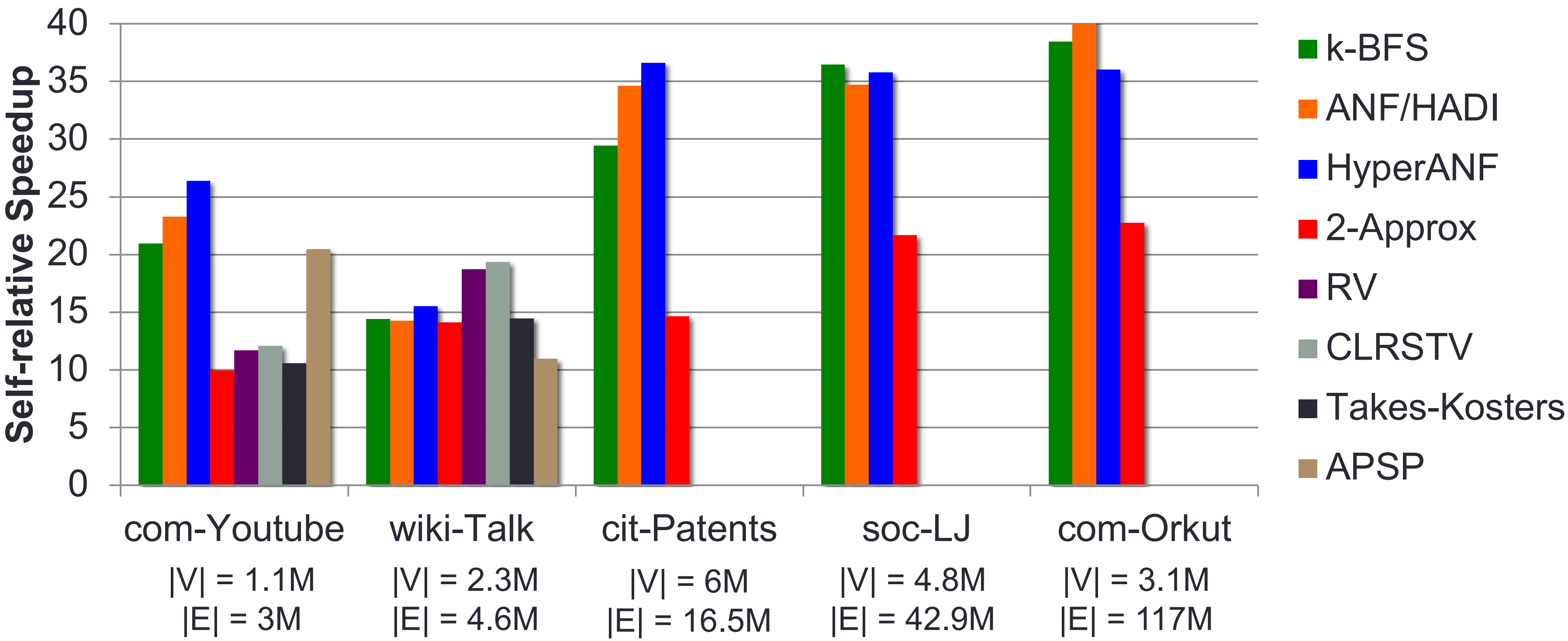
Performance on Synthetic Graphs





Parallel Scalability

Experiments done on a 40-core Intel Nehalem machine with 256 GB memory

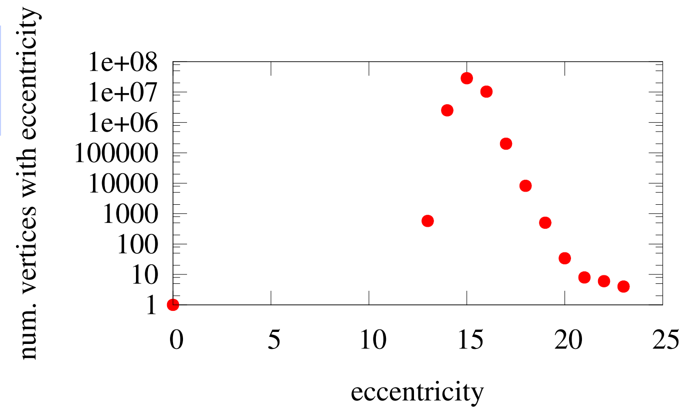


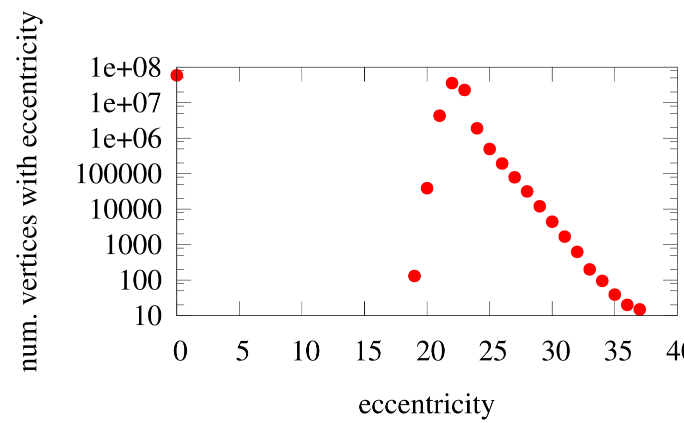
- k-BFS achieves 14—38x speedup on 40 cores (higher for larger graphs)

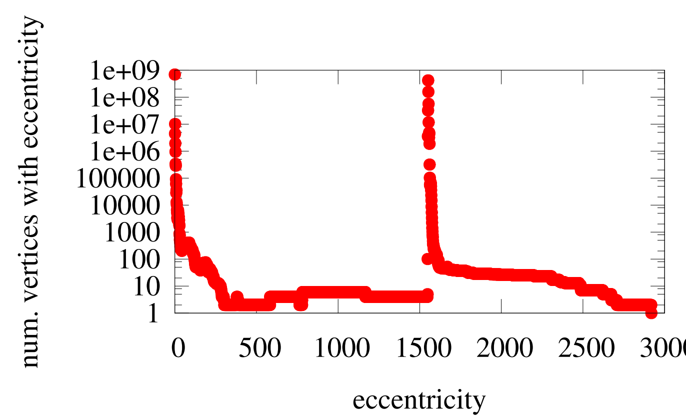
Scaling to Large Graphs

Experiments done on a 40-core Intel Nehalem machine with 256 GB memory

Graph	Size	k-BFS (k=64) Time
Twitter	$\lvert V\rvert = 4.2 \times 10^7$ $\lvert E\rvert = 1.2 \times 10^9$	3.5 minutes
Friendster	$\lvert V\rvert = 1.2 \times 10^8$ $\lvert E\rvert = 1.8 \times 10^9$	1.5 minutes
Yahoo!	$\lvert V\rvert = 1.4 \times 10^9$ $\lvert E\rvert = 6.4 \times 10^9$	11 minutes







Conclusion

- **Comprehensive evaluation** of shared-memory parallel eccentricity estimation algorithms
- k-BFS is orders of magnitude more **accurate** for a fixed running time than other estimation algorithms
- k-BFS is **scalable** to largest publicly-available real-world graphs studied in the literature

- Future work
  - Extensions to directed, weighted graphs
  - Theoretical bounds for (variants of) k-BFS

Thank you!

Code: http://github.com/jshun/ligra