MapReduce Algorithmics

Sergei Vassilvitskii

Yahoo! Research

Based on work with: Bahman Bahmani, Howard Karloff, Ravi Kumar, Silvio Lattanzi, Ben Moseley, Siddharth Suri, Andrea Vattani

Dealing With Massive Data

Dealing With Massive Data

	Memory	
	Polynomial	Sublinear
	RAM	Streaming External Memory Property Testing

Dealing With Massive Data

		Memory: Polynomial	Memory: Sublinear
Processors	Single	RAM	Streaming External Memory Property Testing
Processors	Multiple	PRAM	

Dealing With Massive Data

Processors \ Memory	Polynomial	Sublinear
Single	RAM	Streaming External Memory Property Testing
Multiple	PRAM	MapReduce Distributed Sketches

Modeling MapReduce

Modeling MapReduce

Memory

- Typical datasets 100Gb+
- Cannot store the data in memory
- Insist on sublinear memory

Modeling MapReduce

Memory

- Typical datasets 100Gb+
- Cannot store the data in memory
- Insist on sublinear memory

Machines

- Machines in a cluster do not share memory
- Shared clusters have 100-1000 machines
- Insist on sublinear number of machines

Modeling MapReduce

Memory

- Typical datasets 100Gb+
- Cannot store the data in memory
- Insist on sublinear memory

Machines

- Machines in a cluster do not share memory
- Shared clusters have 100-1000 machines
- Insist on sublinear number of machines

Synchronization

- Computation proceeds in rounds
- Count the number of rounds

Not Modeling MapReduce

Lies, Damned Lies, Statistics

- And big-O notation
- And Competitive Analysis
- And...

Not Modeling MapReduce

Lies, Damned Lies, Statistics

- And big-O notation
- And Competitive Analysis
- And...

MapReduce Communication:

- Very important, makes a big difference

Not Modeling MapReduce

Lies, Damned Lies, Statistics

- And big-O notation
- And Competitive Analysis
- And...

MapReduce Communication:

- Very important, makes a big difference
- Many engineering improvements:
  - Dealing with Graphs: save graph structure locally between rounds
  - Move code to data (and not data to code)
  - Job scheduling (same rack / different racks, etc)

Algorithmics

Algorithmics

Filering:

- Reduce the problem size in parallel
- Solve the smaller instance sequentially

Algorithmics

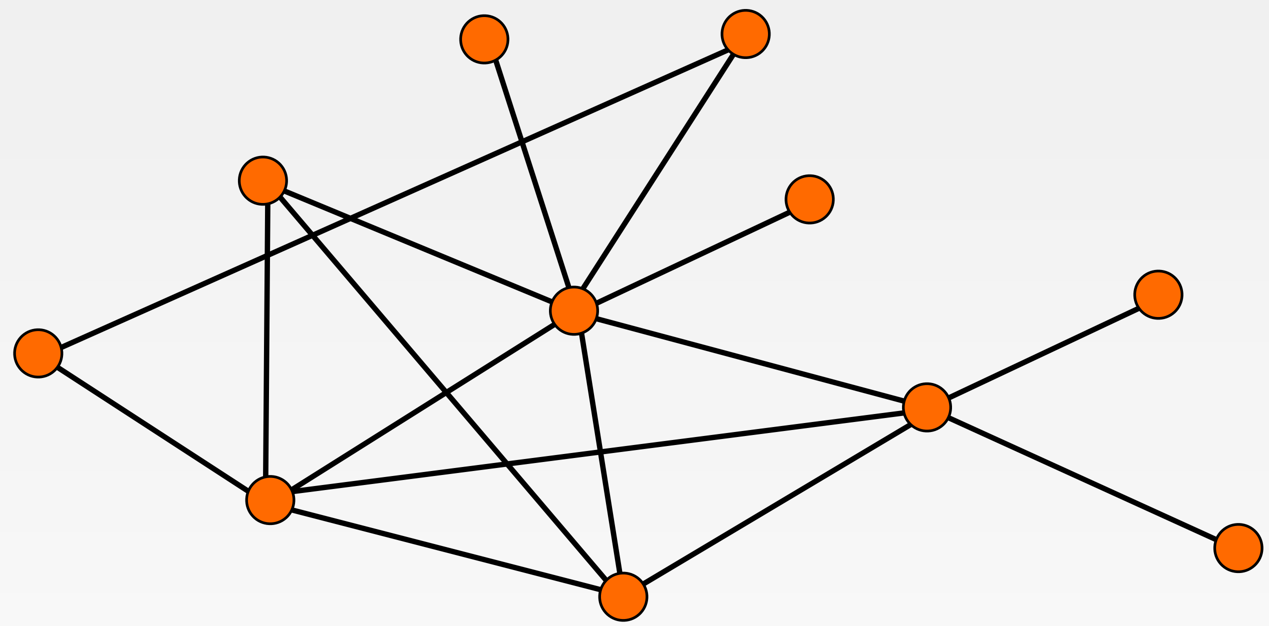
Filtering:

- Reduce the problem size in parallel
- Solve the smaller instance sequentially

How to reduce input size?

- Connectivity: if (u,v) already connected, remove edge
- MST: remove heaviest edge on every cycle
- Matching: remove dead edges (see next talk)
- Clustering: remove nodes that are not in the coreset (see Ben's talk)
- Set Cover: remove dominated sets
- etc

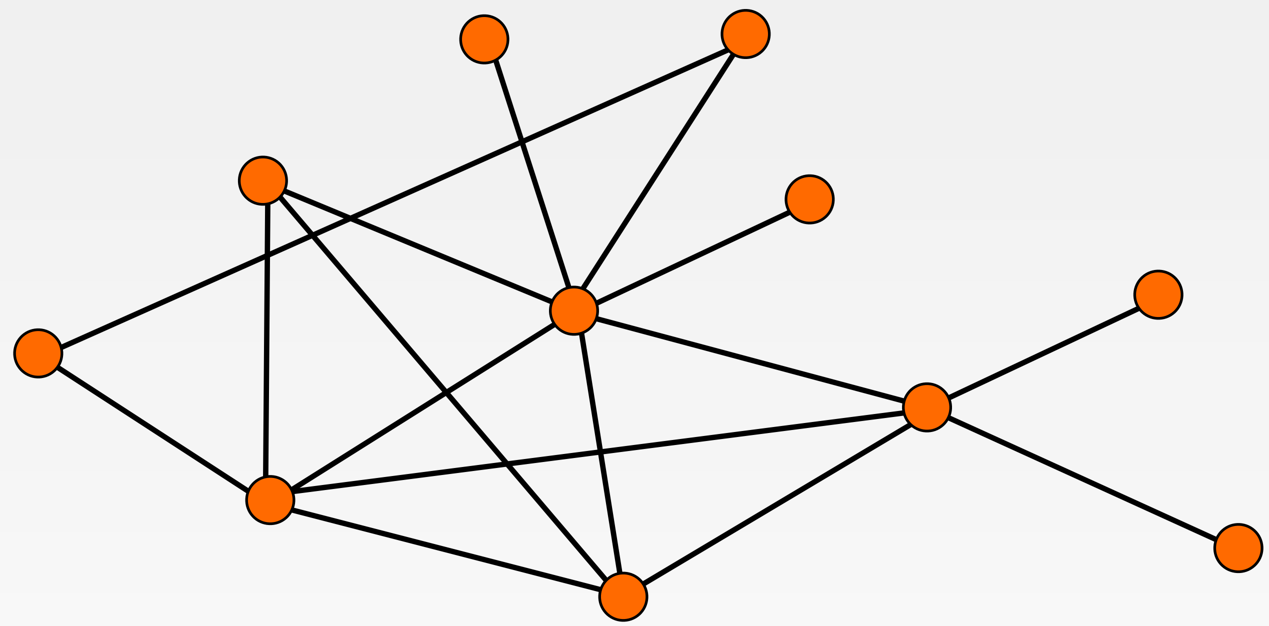
Finding Densest Subgraph



Problem: Given a graph $G = (V, E)$, find $V' \subseteq V$ that maximizes:

$$\rho = \frac{|E(V')|}{|V'|}$$

Finding Densest Subgraph



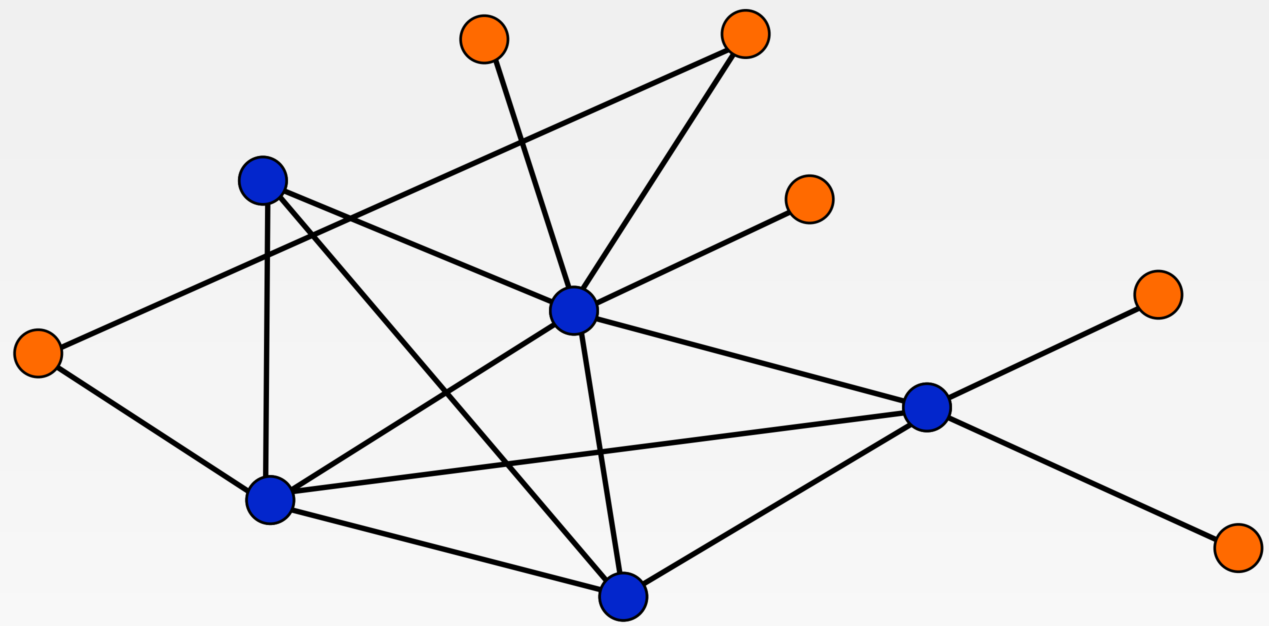
Problem: Given a graph $G = (V, E)$, find $V' \subseteq V$ that maximizes:

$$\rho = \frac{|E(V')|}{|V'|}$$

Useful Primitive in Graph Analysis:

- Community Detection
- Graph Compression
- Link SPAM Mining
- Many other applications

Finding Densest Subgraph



Problem: Given a graph $G = (V, E)$, find $V' \subseteq V$ that maximizes:

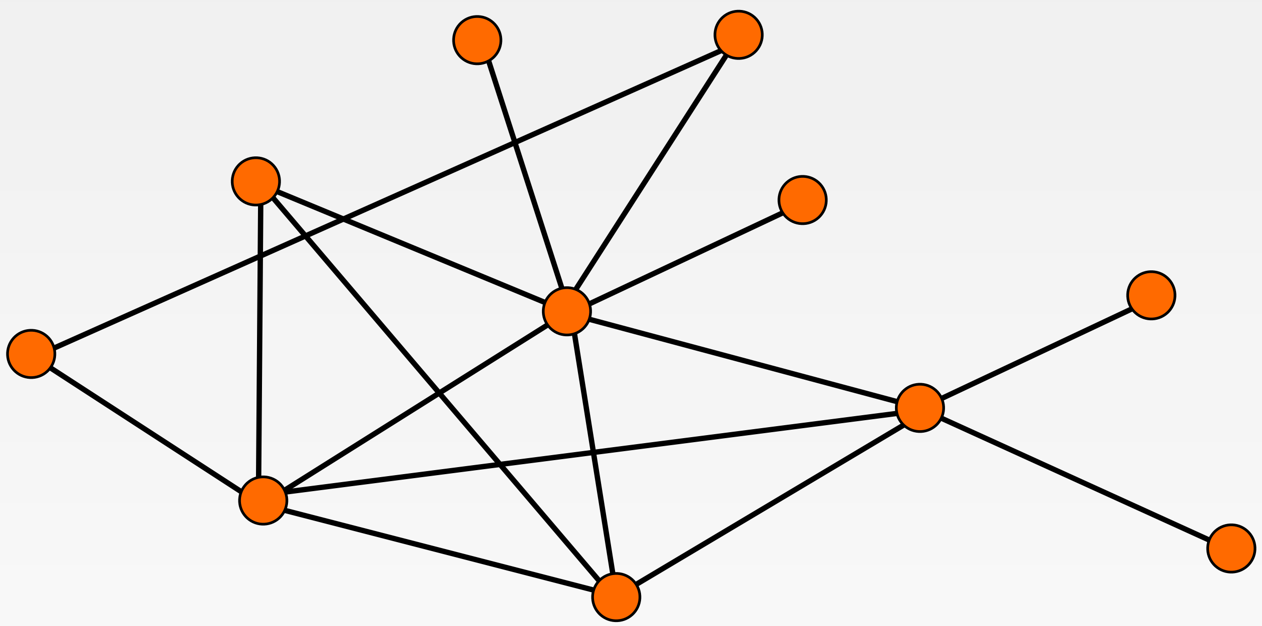
$$\rho = \frac{|E(V')|}{|V'|}$$

Useful Primitive in Graph Analysis

Can be solved exactly:

- LP Formulation
- Multiple Max flow computations

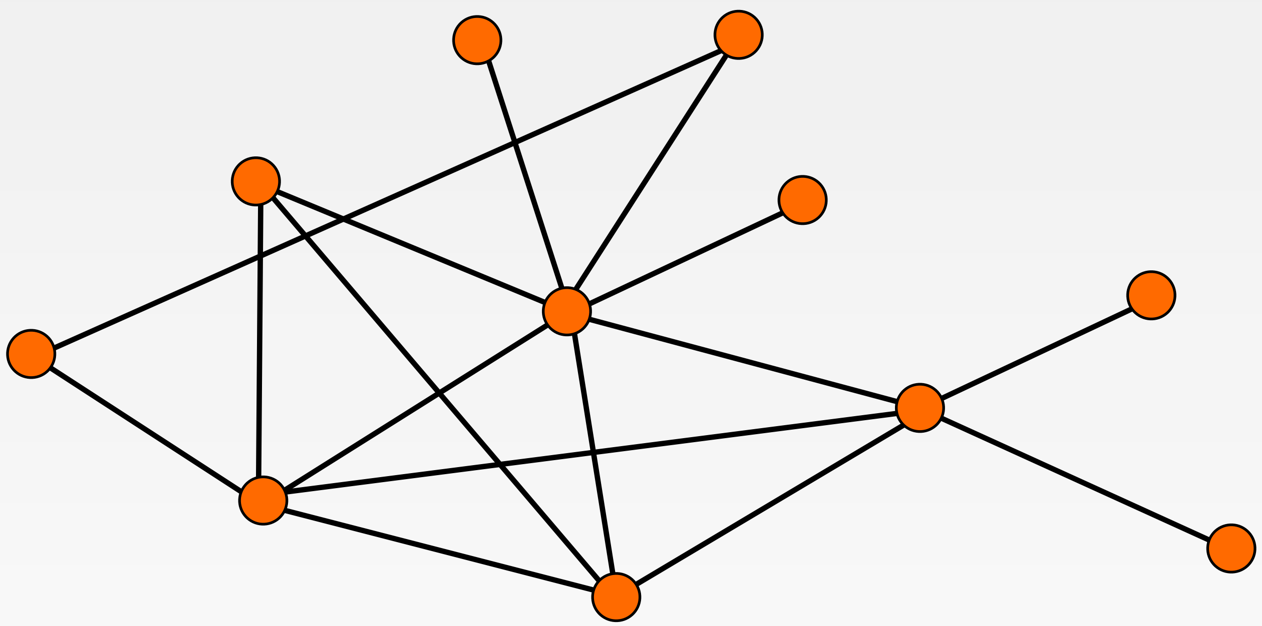
Finding Densest Subgraphs



Simple Algorithm [Charikar ’00]:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

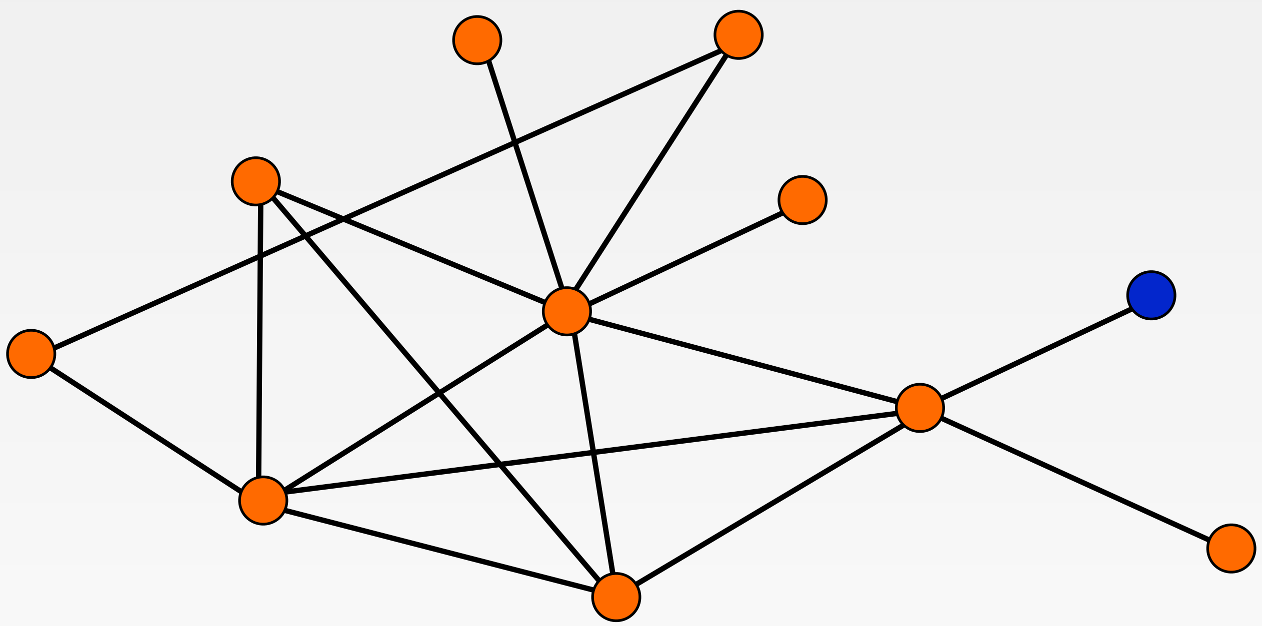


```
Best Density: 16/11
Current Density: 16/11
```

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

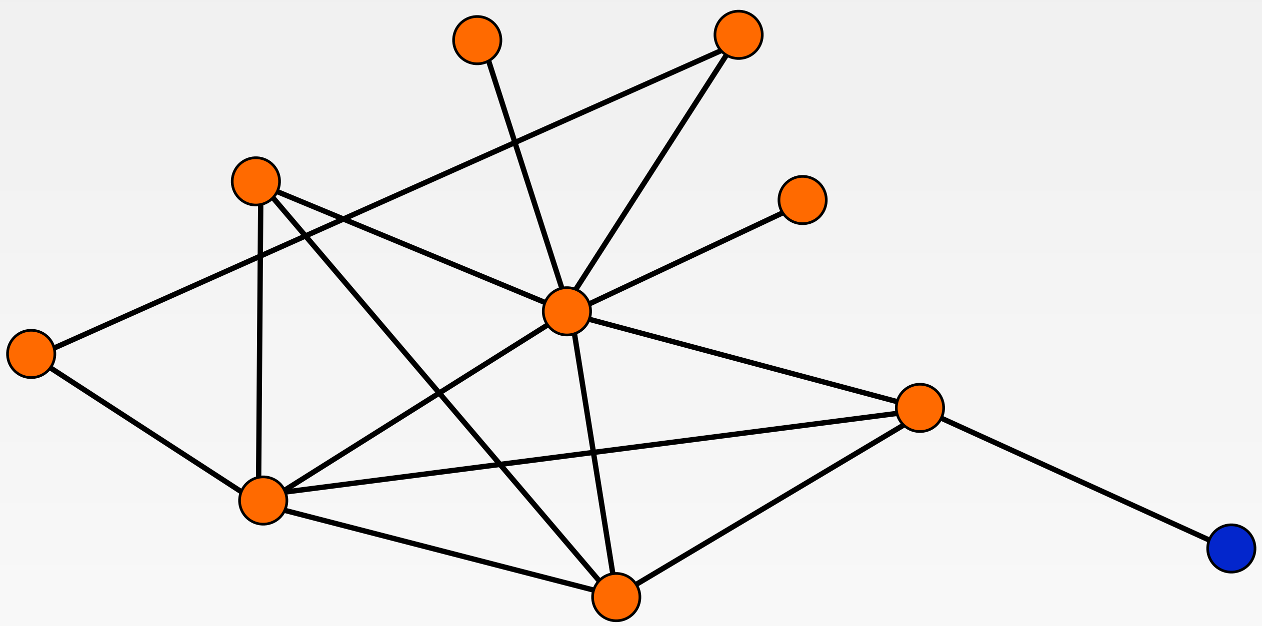


```
Best Density: 16/11
Current Density: 16/11
```

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

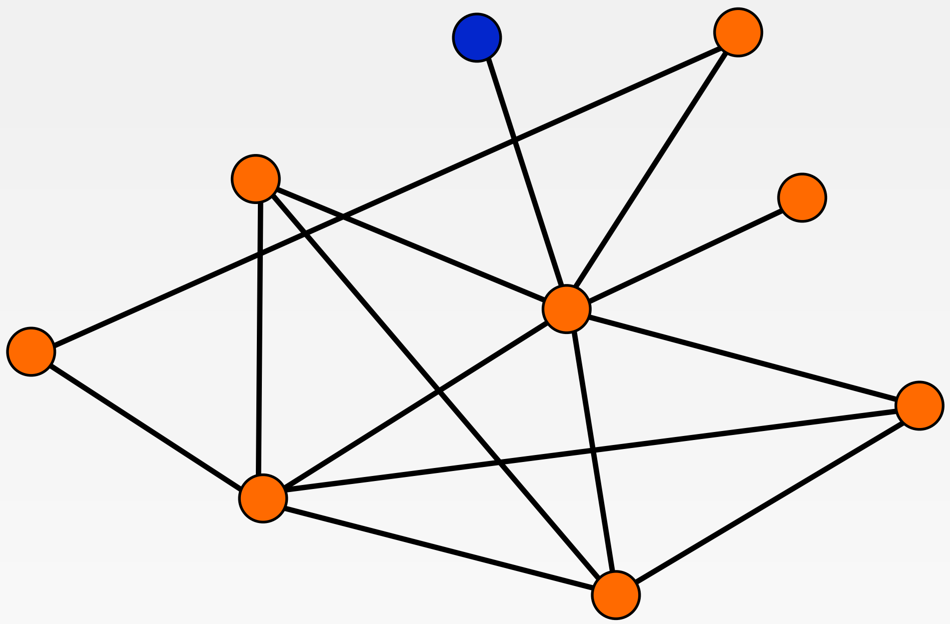


```
Best Density: 15/10
Current Density: 15/10
```

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

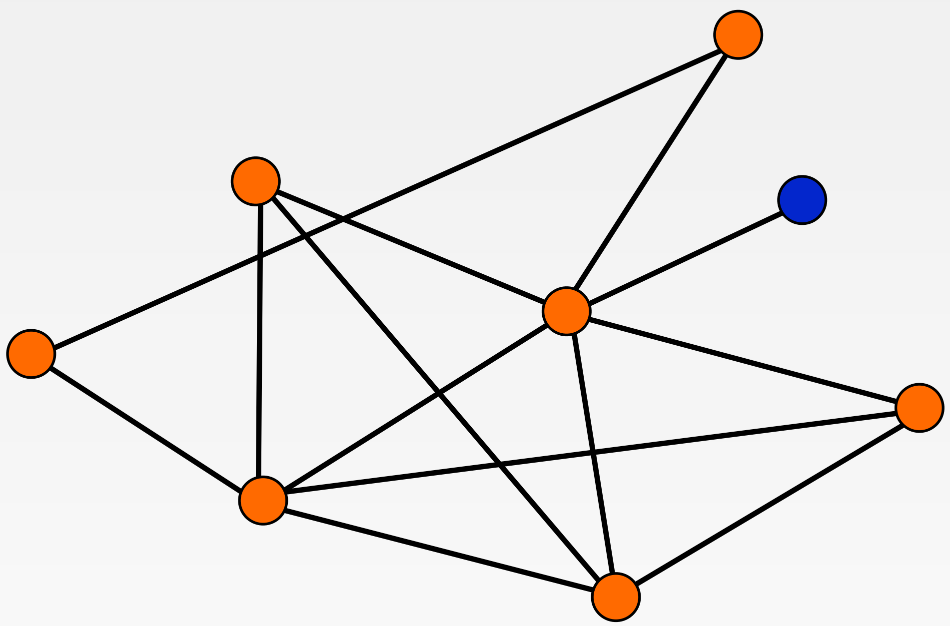


```
Best Density: 14/9
Current Density: 14/9
```

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

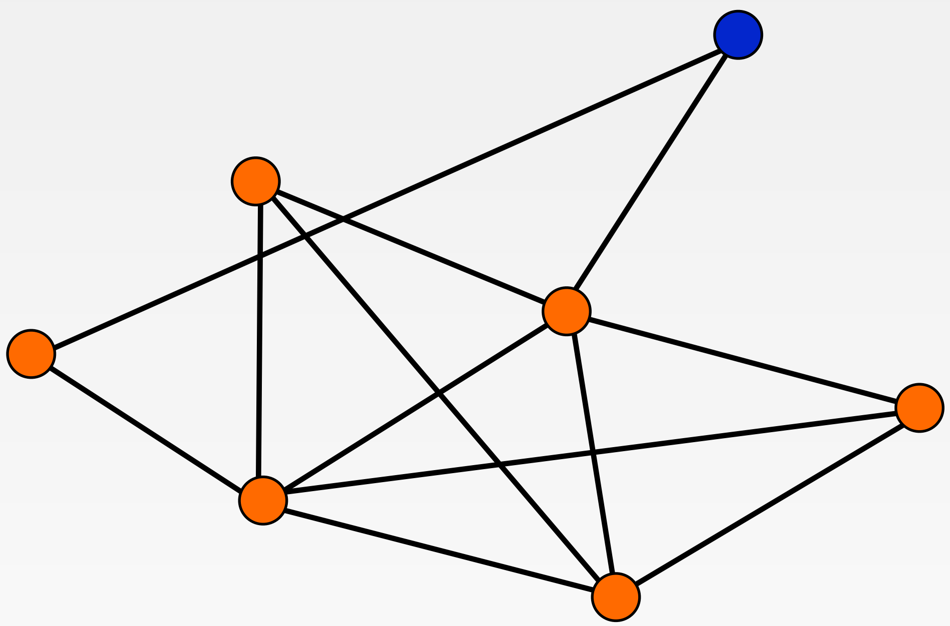


```
Best Density: 13/8
Current Density: 13/8
```

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs



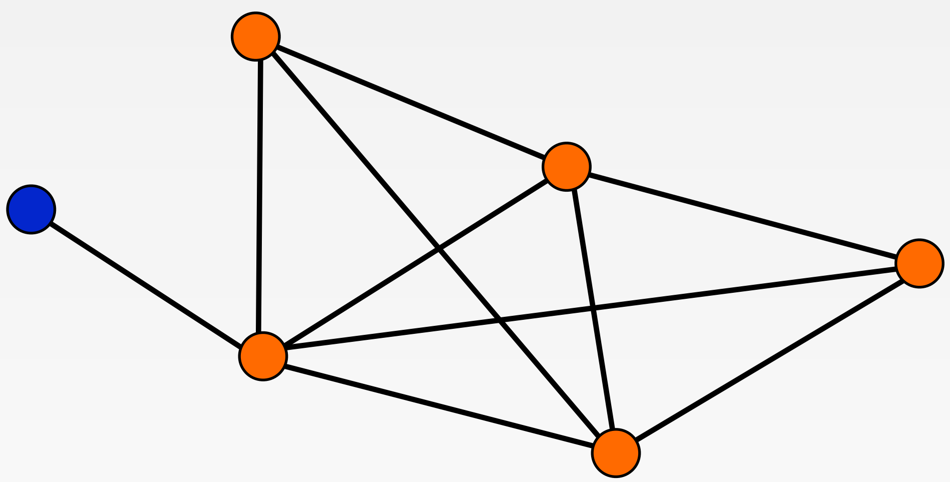
Best Density: 12/7

Current Density: 12/7

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

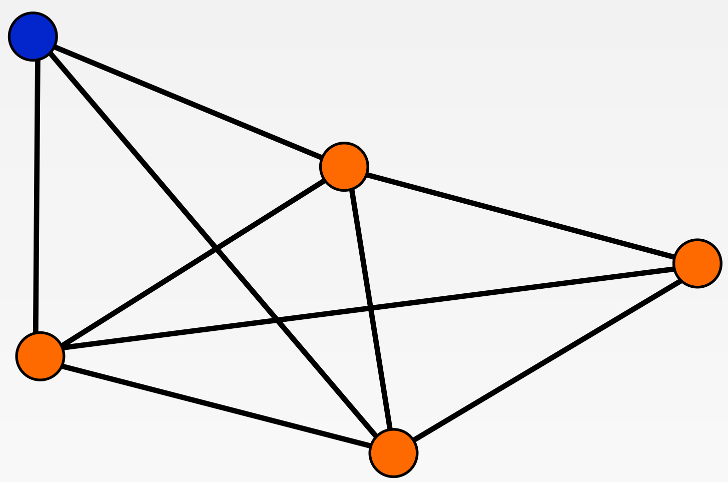


```
Best Density: 12/7
Current Density: 10/6
```

Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs



Best Density: 9/5

Current Density: 9/5

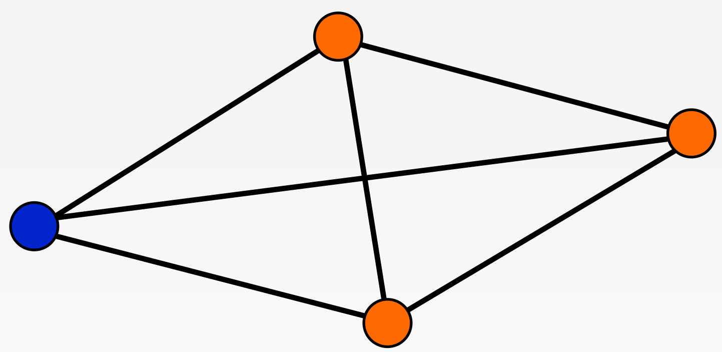
Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

Best Density: 9/5

Current Density: 6/4



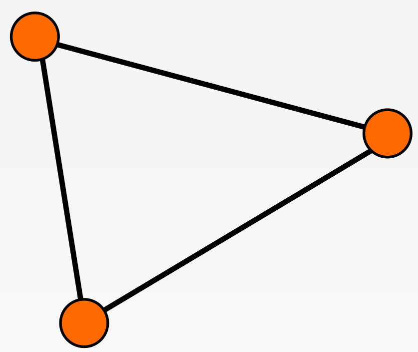
Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs

Best Density: 9/5

Current Density: 3/3



Simple Algorithm:

- Iteratively remove the lowest degree node and update vertex degrees
- Keep the densest intermediate subgraph

Finding Dense Subgraphs (Analysis)

Approximation Ratio:

- Guaranteed to return a 2-approximation

Proof:

- Let $V^* \subseteq V$ be the optimal solution, and $\lambda^* = \frac{|E[V^*]|}{|V^*|}$ the optimal density.
- Consider the first time a vertex from V^* is removed.
- Every vertex in V^* has degree at least λ^*.
  - Otherwise can improve optimum density
- Therefore the density of that subgraph is at least:

$$\frac{\lambda^* |V^*|}{2|V^*|} = \lambda^*/2$$

Finding Dense Subgraphs (Analysis)

Approximation Ratio:

- Guaranteed to return a 2-approximation

Running Time:

- RAM:
  - Maintain a heap on vertex degrees
  - Update keys upon removing every edge
  - Straightforward implementation in $O(m \log n)$
- Streaming:
  - Seemingly need one pass per vertex to adapt this algorithm
  - Can show that need $\Omega(n/\log n)$ memory if using $O(\log n)$ passes
- MapReduce?
  - Open question in Chierichetti, Kumar and Tompkins WWW '10.

Parallel Dense Subgraphs

Sequential Algorithm:

- Remove the node with the smallest degree

Parallel Dense Subgraphs

Sequential Algorithm:

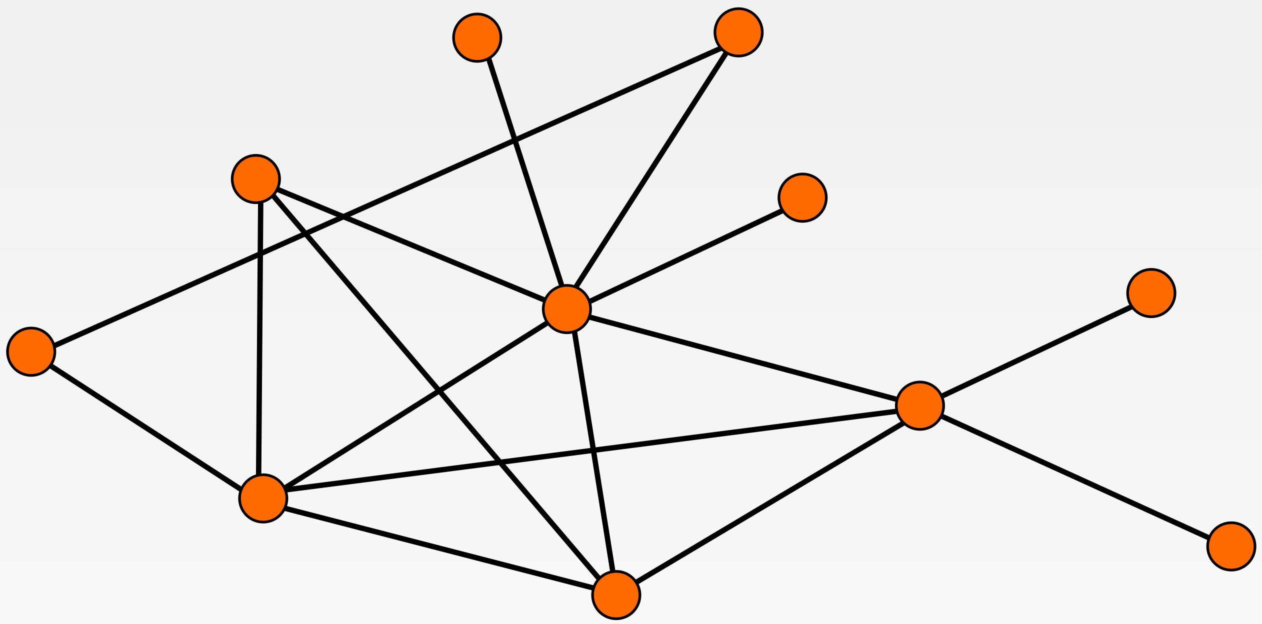
- Remove the node with the smallest degree

Parallel Version:

- Remove all nodes with less degree less than $(1+\epsilon)$ * average degree
- Of course this also includes the smallest degree node

- Every Step:
  - Round 1: Count remaining edges, vertices, compute vertex degrees
    - Distributed counting
  - Round 2: Remove vertices with degree below threshold
    - Distributed checking

Parallel Dense Subgraphs

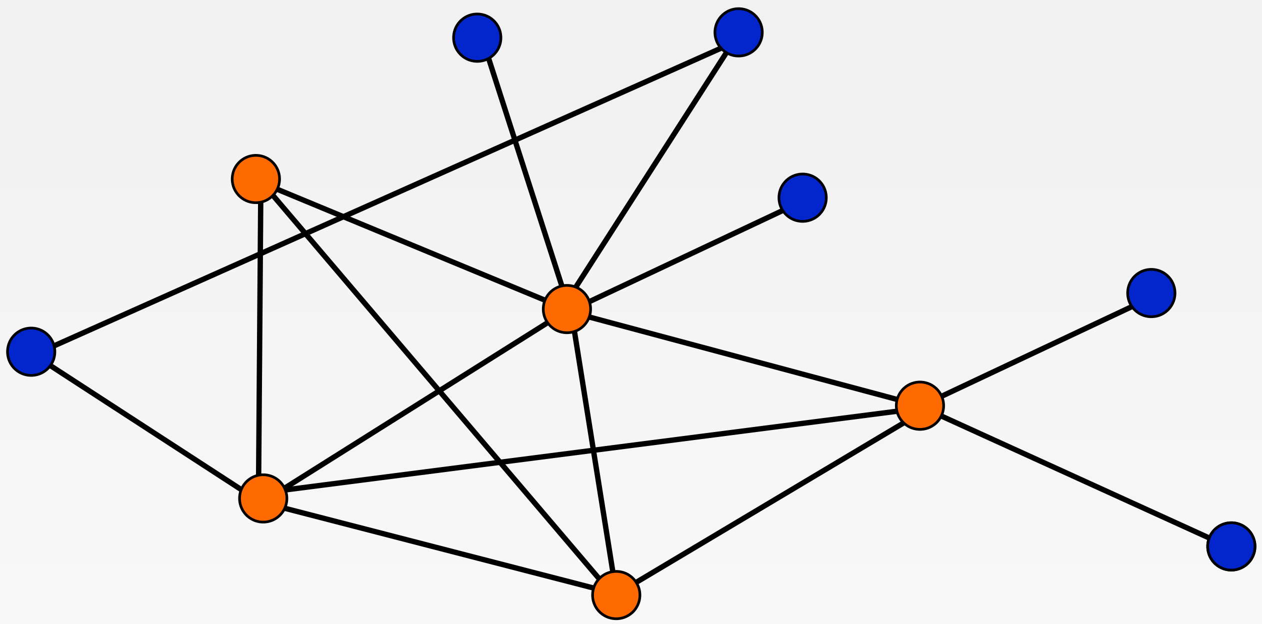


```
Best Density: 16/11
Current Density: 16/11
Average Degree: 32/11
```

Parallel Algorithm:

- Iteratively remove nodes with degree below average and update vertex degrees
- Keep the densest intermediate subgraph

Parallel Dense Subgraphs

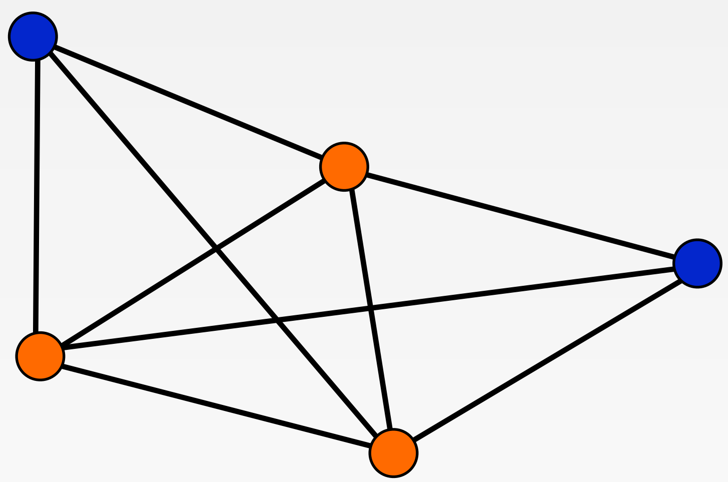


```
Best Density: 16/11
Current Density: 16/11
Average Degree: 32/11
```

Parallel Algorithm:

- Iteratively remove nodes with degree below average and update vertex degrees
- Keep the densest intermediate subgraph

Parallel Dense Subgraphs

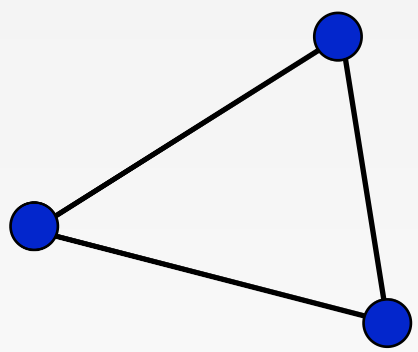


```
Best Density: 9/5
Current Density: 9/5
Average Degree: 18/5
```

Parallel Algorithm:

- Iteratively remove nodes with degree below average and update vertex degrees
- Keep the densest intermediate subgraph

Parallel Dense Subgraphs



```
Best Density: 9/5
Current Density: 3/3
Average Degree: 6/3
```

Parallel Algorithm:

- Iteratively remove nodes with degree below average and update vertex degrees
- Keep the densest intermediate subgraph

Parallel Densest Subgraph (Analysis)

Algorithm:

- Each round remove all vertices with degree less than $(1+\epsilon)$ * average.

How many vertices do we remove?

- One cannot have too many vertices above average (This is not Lake Wobegon)
- Easy [Markov inequality] : at most a $\frac{1}{1+\epsilon}$ fraction of vertices remains in every round.
- Therefore algorithm terminates after $O\left(\frac{1}{\epsilon}\log n\right)$ rounds

Parallel Densest Subgraph (Analysis)

Algorithm:

- Each round remove all vertices with degree less than $(1+\epsilon)$ * average.

How many vertices do we remove?

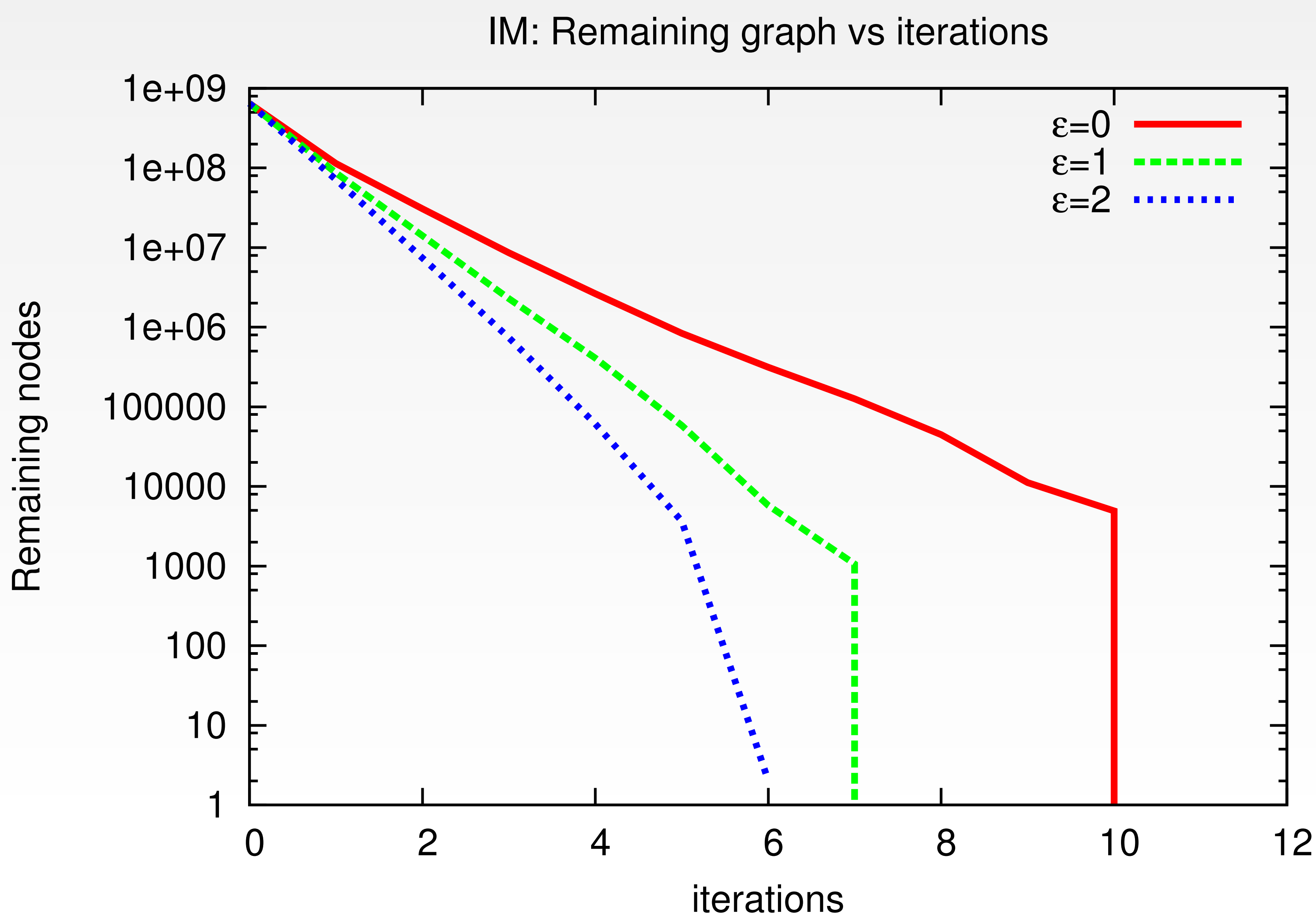
- One cannot have too many vertices above average (This is not Lake Wobegon)
- Easy [Markov inequality] : at most a $\frac{1}{1+\epsilon}$ fraction of vertices remains in every round.
- Therefore algorithm terminates after $O\left(\frac{1}{\epsilon}\log n\right)$ rounds

Approximation Ratio:

- Achieves a $(2+\epsilon)$ approximation in the worst case
  - Only look at the degree of the nodes removed as compared to average. in

How well does it work?

IM Network graph: 650M nodes, 6.1B edges



- Quickly reduce the size of the graph.
- Approximation ratio between 1.06 and 1.4 at $\epsilon = 1$

Improving Sequential Algorithms

Densest Subgraph

- Original algorithm: O(m) heap updates:
  - Update vertex degrees every time an edge is removed.
- New algorithm O(n) heap updates:
  - Number of vertices decreases geometrically every round

Improving Sequential Algorithms

Low Memory Algorithms:

- Recall MapReduce requirement of sublinear memory
- Can run the parallel algorithm sequentially
  - Work efficient algorithms imply identical running time

Improving Sequential Algorithms

Low Memory Algorithms:

- Recall MapReduce requirement of sublinear memory
- Can run the parallel algorithm sequentially
  - Work efficient algorithms imply identical running time

In practice:

- Low memory algorithms are more efficient
- Take better advantage of caching hierarchy (L1, L2, OS)
- Empirically have observed faster running times running MapReduce algorithms sequentially

Wrapping Up

Conclusion:

- MapReduce combines parallelism with sublinear memory
- Filtering:
  - Reduce input size in parallel
  - Until data is small enough to be processed sequentially
- Unlike PRAMs, insisting on non-shared memory leads to very good cache performance when simulating sequentially.

Thank You

sergei@yahoo-inc.com