MapReduce and Distributed Data Analysis

Sergei Vassilvitskii

Google Research

Dealing With Massive Data

Dealing With Massive Data

	Memory	
	Polynomial	Sublinear
	RAM	Sketches External Memory Property Testing

Dealing With Massive Data

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

Dealing With Massive Data

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

What about Data Streams?

	Input	
	Batch	Online
	RAM	Data Streams

What about Data Streams?

Processors \ Input	Batch	Online
Single	RAM	Data Streams
Multiple	MapReduce	Distributed Sketches Active DHTs

Today's Focus: MapReduce

Multiple Processors:

- 10s to 10,000s processors

Sublinear Memory

- A few Gb of memory/machine, even for Tb+ datasets
- Unlike PRAMs: memory is not shared

Batch Processing

- Analysis of existing data
- Extensions used for incremental updates, online algorithms

Why MapReduce?

Practice:

- Used very widely for large data analysis
- Google, Yahoo!, Amazon, Facebook, Netflix, LinkedIn, New York Times, eHarmony, ...

Is it a Fad?:

- No! (Do Fads last 10 years?)
- Many similar implementations and abstractions on top of MR: Hadoop, Pig, Hive, Flume, Pregel, ...
- Same computational model underneath

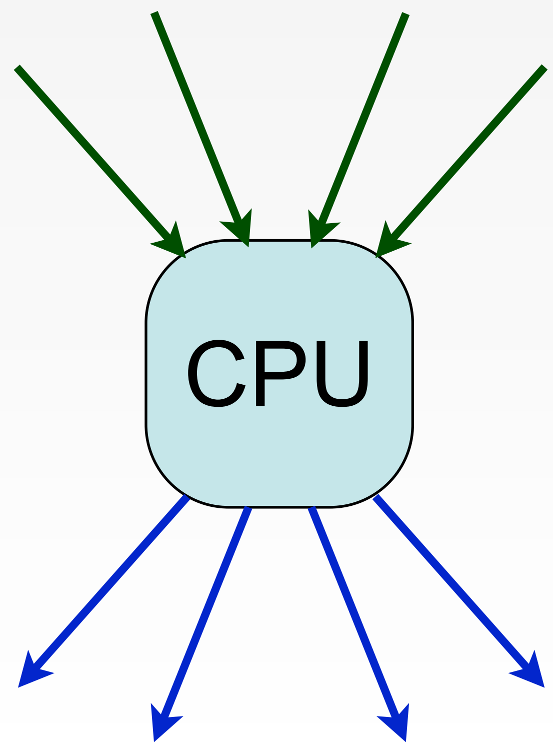
Why?

- Practice that needs theory
- Makes data locality explicit (which sometimes leads to faster sequential algorithms) !

MR Flow

Computation proceeds in rounds.

Each round:



Receive Input

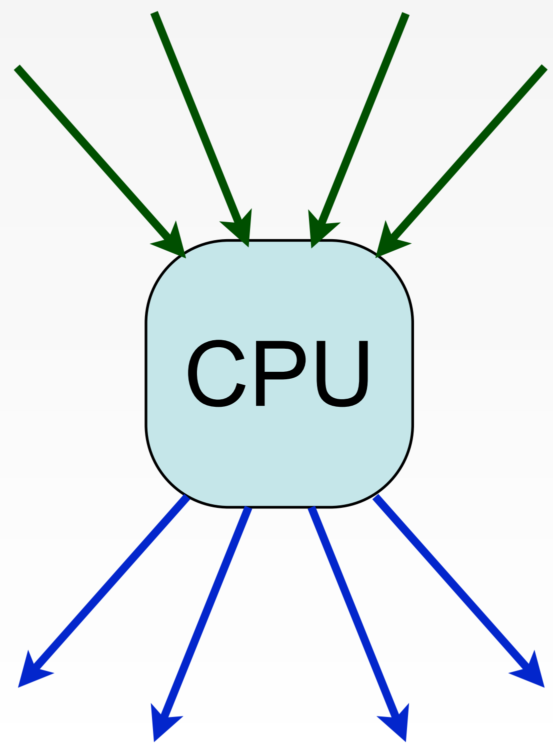
Compute

Produce output

MR Flow

Computation proceeds in rounds.

Each round:



Receive Input: Input must fit into memory

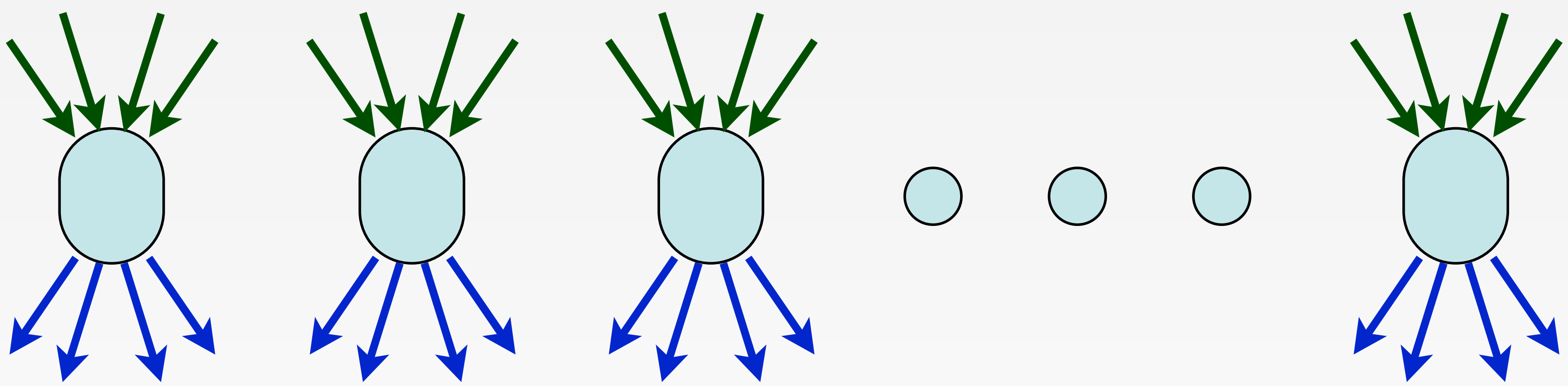
Compute: Full TM

Produce output: Annotated with address of machine in next round

MR Multiple Rounds

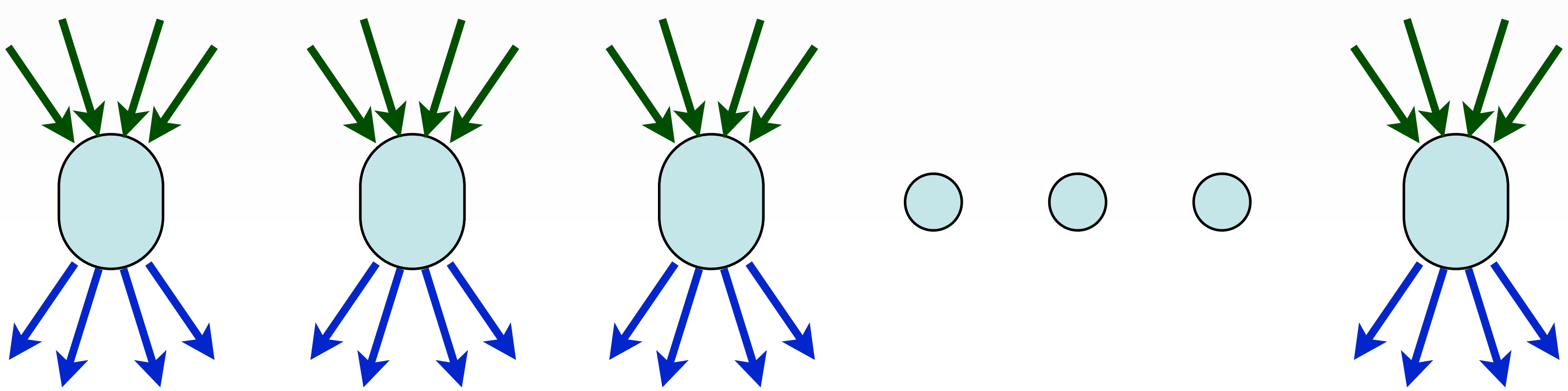
Many Machines...and multiple rounds

Round 1:



Arbitrary communication topology (depends on the output of round 1)

Round 2:



Data Streams vs. MapReduce

Distributed Sum:

- Given a set of n numbers: $a_1, a_2, \ldots, a_n \in \mathbb{R}$, find $S = \sum_i a_i$

Stream:

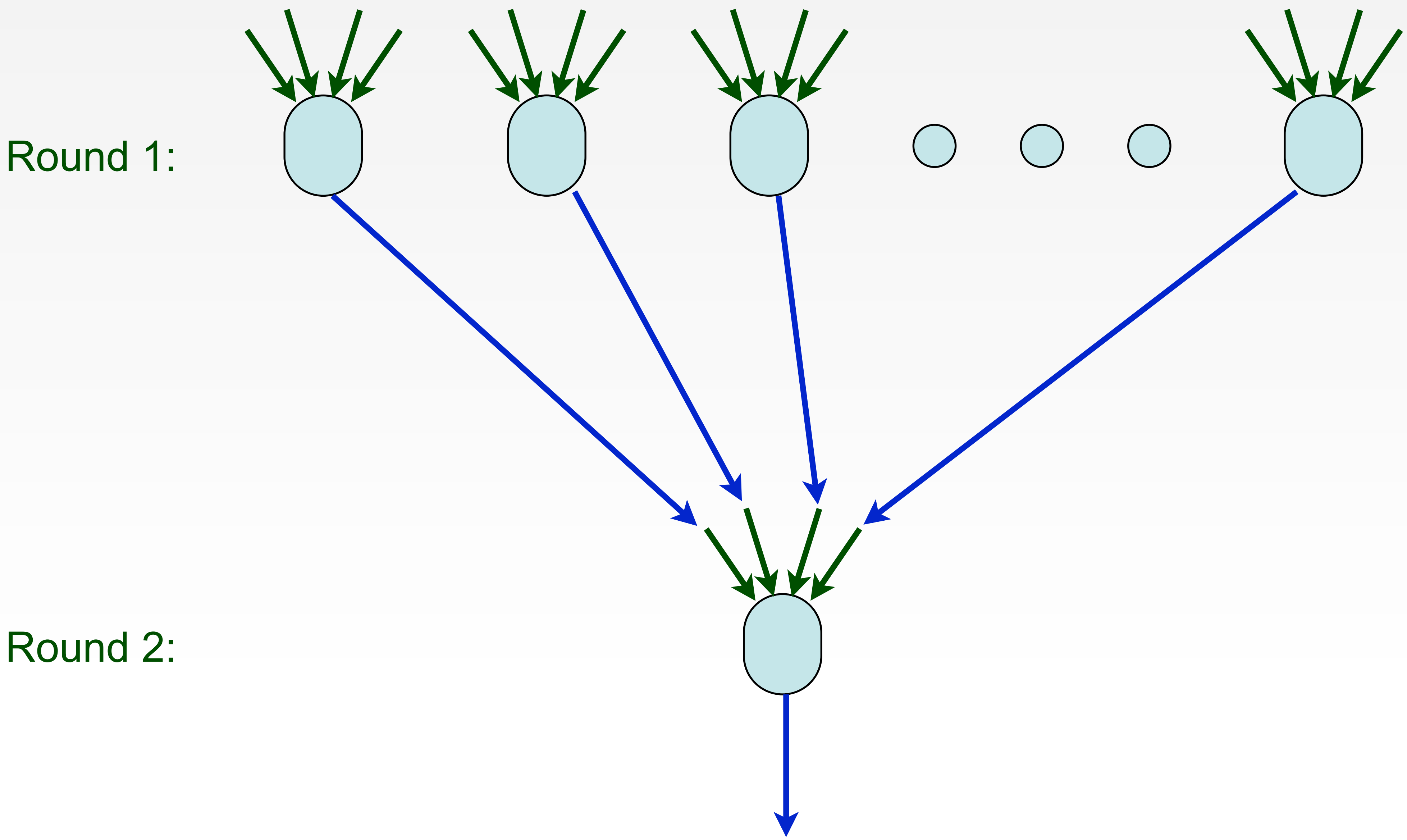
- Maintain a partial sum $S_j = \sum_{i \leq j} a_i$
- update with every element

MapReduce:

- Compute $M_j = a_{jk} + a_{jk+1} + \ldots + a_{j(k+1)-1}$ for $k = \sqrt{n}$ in Round 1
- Round 2: add the $\sqrt{n}$ partial sums.

MR Modeling [Sketch]

Distributed Sum in MR



Modeling

For an input of size n :

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $n^{1-\epsilon}$ for some $\epsilon > 0$

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $n^{1-\epsilon}$ for some $\epsilon > 0$

Machines

- Machines in a cluster do not share memory
- Insist on sublinear number of machines: $n^{1-\epsilon}$ for some $\epsilon > 0$

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $n^{1-\epsilon}$ for some $\epsilon > 0$

Machines

- Machines in a cluster do not share memory
- Insist on sublinear number of machines: $n^{1-\epsilon}$ for some $\epsilon > 0$

Synchronization

- Computation proceeds in rounds
- Count the number of rounds
- Aim for $O(1)$ rounds

Not Modeling

Lies, Damned Lies, Statistics

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

Not Modeling

Lies, Damned Lies, Statistics

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

Communication:

- Very important, makes a big difference

Not Modeling

Lies, Damned Lies, Statistics

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

Communication:

- Very important, makes a big difference
- Order of magnitude improvements due to
  - Move code to data (and not data to code)
  - Working with graphs: save graph structure locally between rounds
  - Job scheduling (same rack / different racks, etc)

Not Modeling

Lies, Damned Lies, Statistics

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

Communication:

- Very important, makes a big difference
- Order of magnitude improvements due to
  - Move code to data (and not data to code)
  - Working with graphs: save graph structure locally between rounds
  - Job scheduling (same rack / different racks, etc)
- Bounded by $n^{2-2\epsilon}$ (total memory of the system) in the model
  - Minimizing communication always a goal

How Powerful is this Model?

Compared to PRAMs:

- Can (formally) simulate PRAM algorithms with MR
- In practice can use same idea without formal simulation
- One round of MR per round of PRAM: $O(\log n)$ rounds total
- Hard to break below $o(\log n)$, need new ideas

How Powerful is this Model?

Different Tradeoffs from PRAM:

- PRAM: LOTS of very simple cores, communication every round
- MR: Many full TM cores, batch communication.

Formally:

- Can simulate PRAM algorithms with MR
- In practice can use same idea without formal simulation
- One round of MR per round of PRAM: $O(\log n)$ rounds total
- Hard to break below $o(\log n)$, need new ideas!

How Powerful is this Model?

Compared to Data Streams:

- Solving different problems (batch vs. online)
- But can use similar ideas (e.g. sketching)

How Powerful is this Model?

Compared to Data Streams:

- Solving different problems (batch vs. online)
- But can use similar ideas (e.g. sketching)

Compared to BSP:

- Closest in spirit
- Do not optimize parameters in algorithm design phase

Outline

Introduction

MapReduce

MR Algorithmics

- Connected Components
- Matchings
- Greedy Algorithms

Open Problems

Algorithmics

Find the core of the problem:

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

Algorithmics

Find the core of the problem:

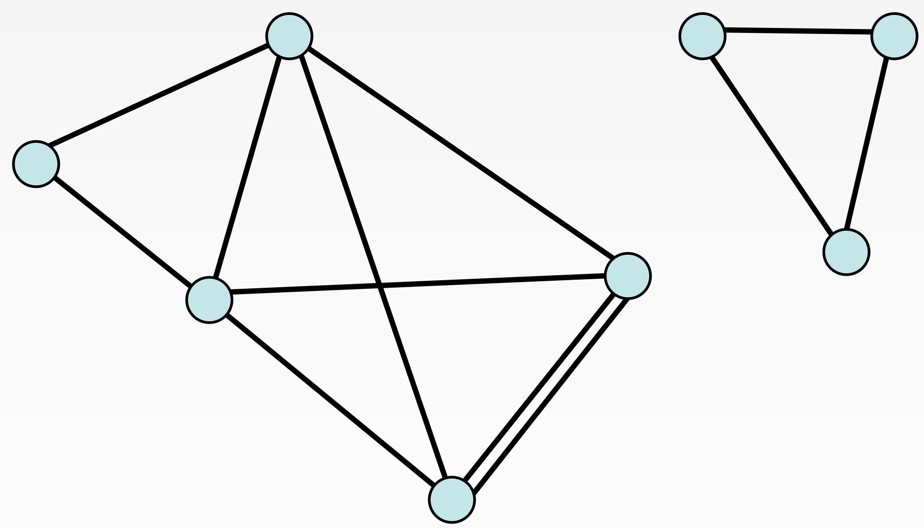
- Reduce the problem size in parallel
- Solve the smaller instance sequentially

Roadmap:

- Identify redundant information
- Filter out redundancy to reduce input size
- Solve the smaller problem
- Use the solution as a seed for the larger problem

Connected Components

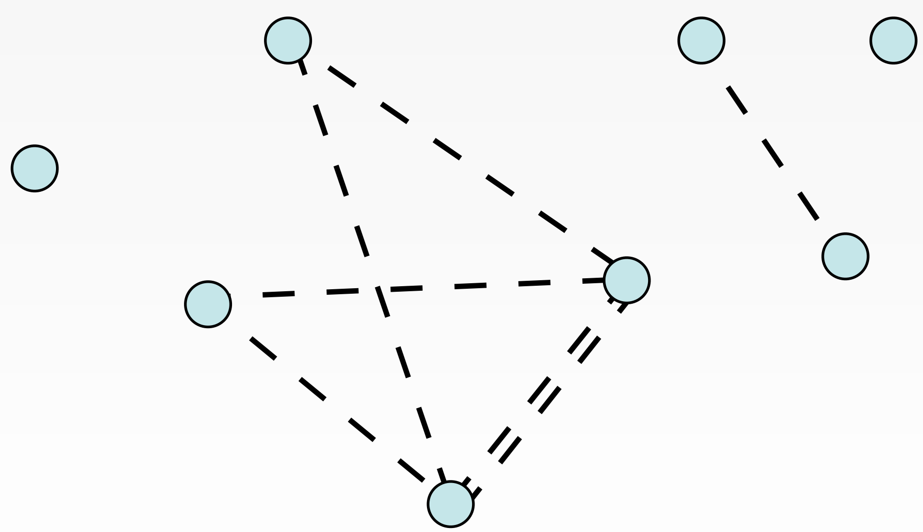
Given a graph:

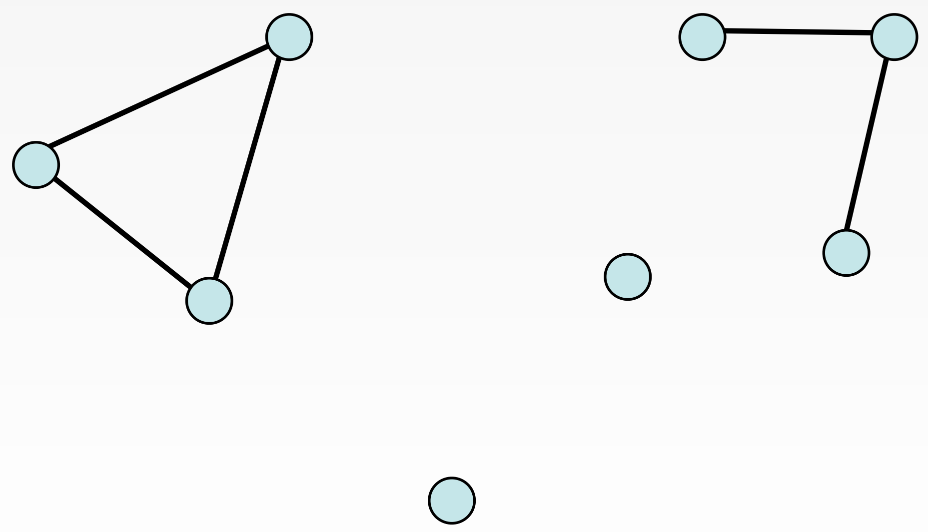


Connected Components

Given a graph:

1. Partition (randomly)





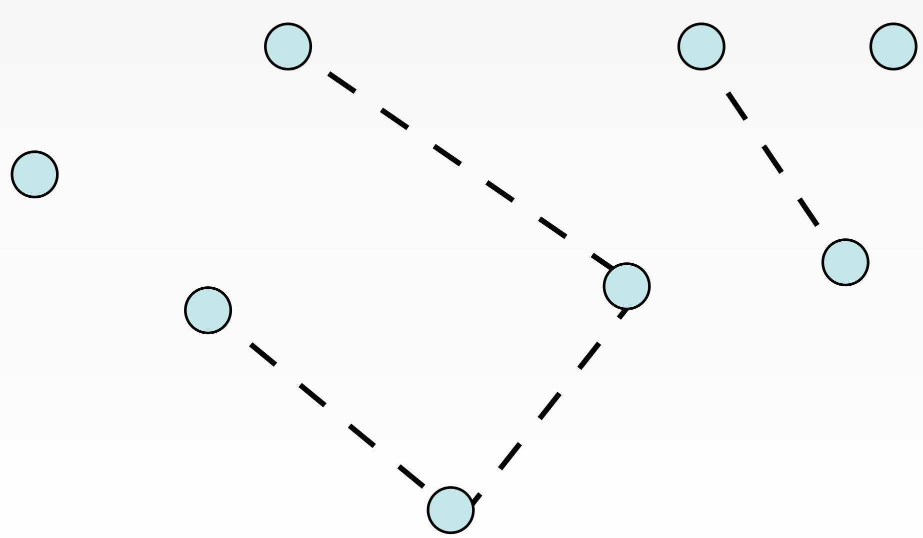
Machine 1

Machine 2

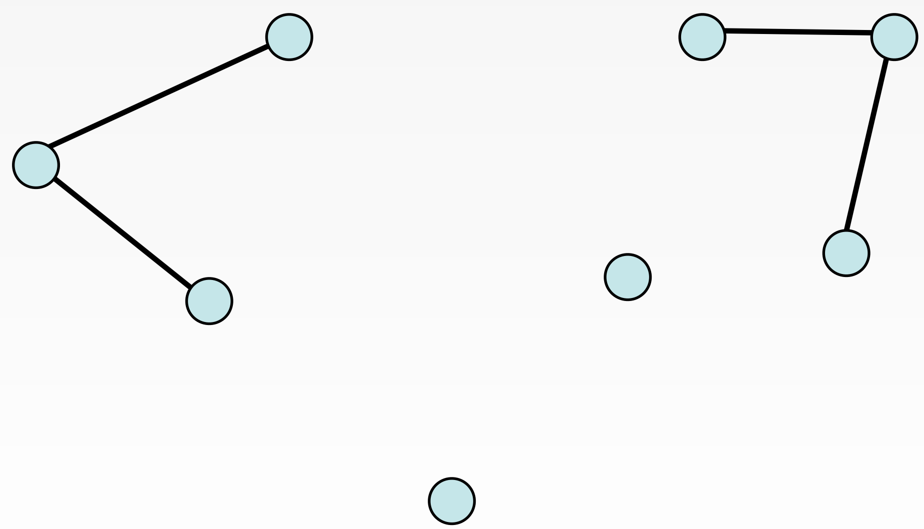
Connected Components

Given a graph:

1. Partition (randomly)
2. Summarize (keep $\leq n-1$ edges per partition)



Machine 1

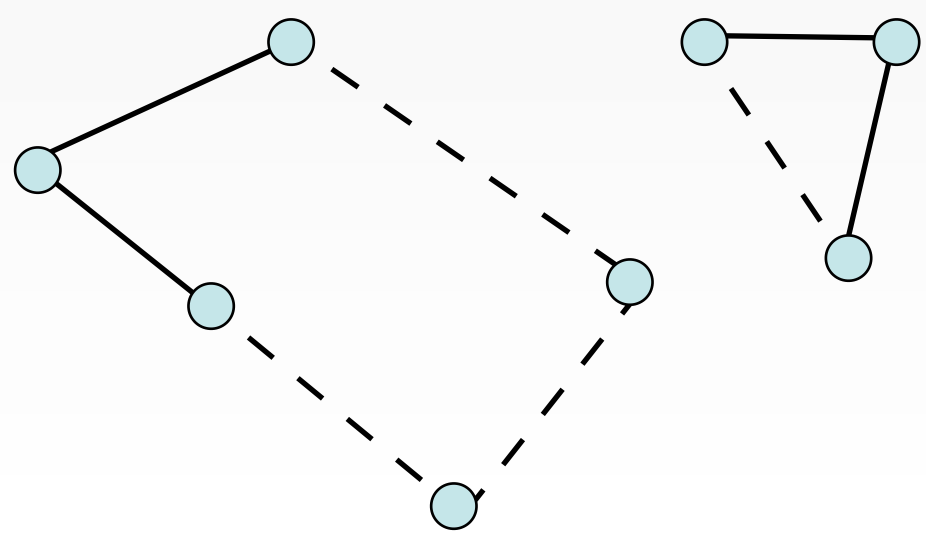


Machine 2

Connected Components

Given a graph:

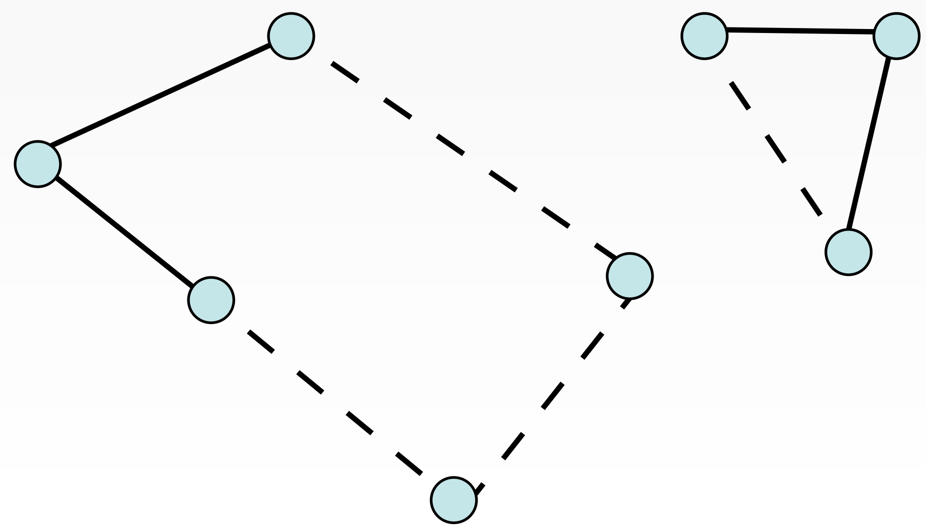
1. Partition (randomly)
2. Summarize (keep $\leq n-1$ edges per partition)
3. Recombine



Connected Components

Given a graph:

1. Partition (randomly)
2. Summarize (keep $\leq n - 1$ edges per partition)
3. Recombine
4. Compute CC's



Analysis

Given: k machines:

- Total Runtime: $O((^m/_k + nk)\alpha(n))$
- Memory per machine: $O(^m/_k + nk)$
  - Actually, can stream through edges so $O(n)$ suffices
- 2 Rounds total

Outline

Introduction

MapReduce

MR Algorithmics

- Connected Components
- Matchings
- Greedy Algorithms

Open Problems

Matchings

Finding matchings

- Given an undirected graph $G = (V, E)$
- Find a maximum matching

Matchings

Finding matchings

- Given an undirected graph $G = (V, E)$
- ~~Find a maximum matching~~
- Find a maximal matching

Matchings

Finding matchings

- Given an undirected graph $G = (V, E)$
- ~~Find a maximum matching~~
- Find a maximal matching

Try random partitions:

- Find a matching on each partition
- Compute a matching on the matchings
- Does not work: may make very limited progress

Looking for redundancy

Matching:

- Could drop the edge if an endpoint already matched

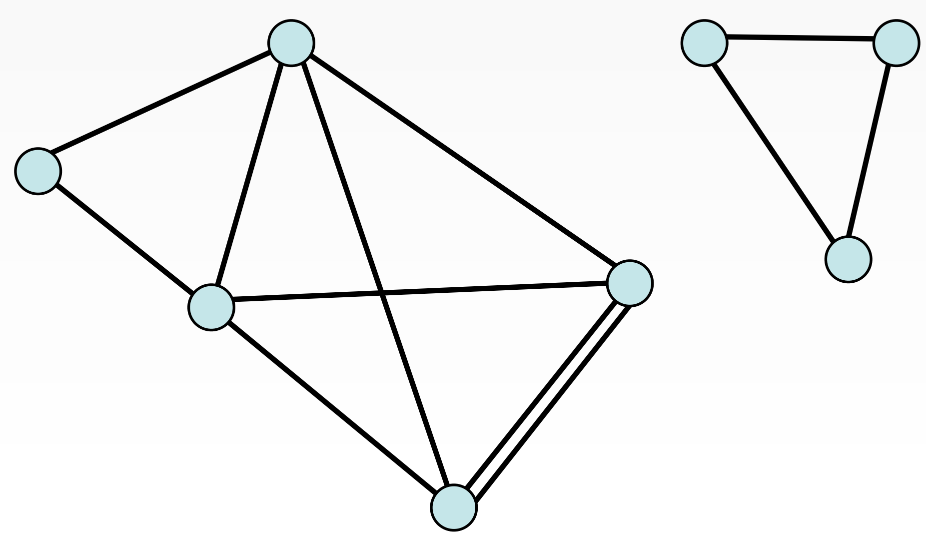
Idea:

- Find a seed matching (on a sample)
- Remove all ‘dead’ edges
- Recurse on remaining edges

Algorithm

Given a graph:

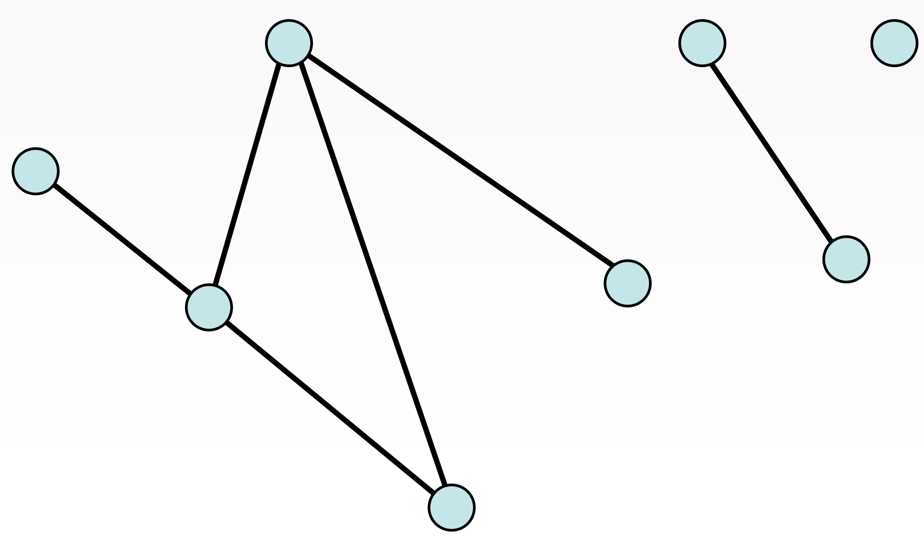
1. Take a random sample



Algorithm

Given a graph:

1. Take a random sample

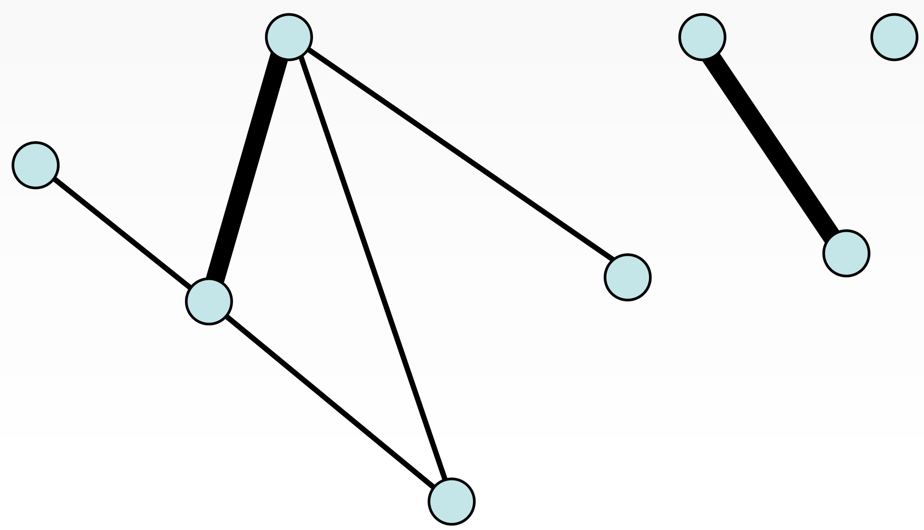


Algorithm

Given a graph:

1. Take a random sample

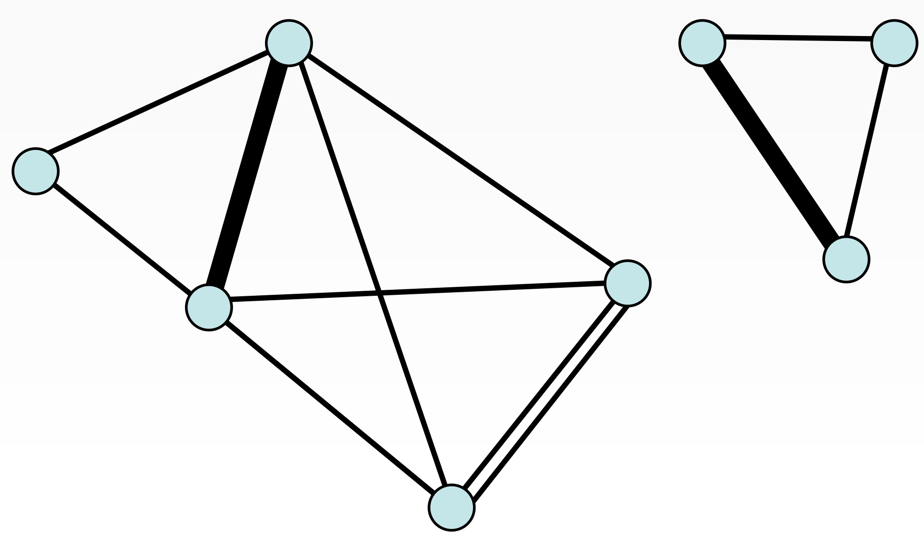
2. Find a maximal matching on sample



Algorithm

Given a graph:

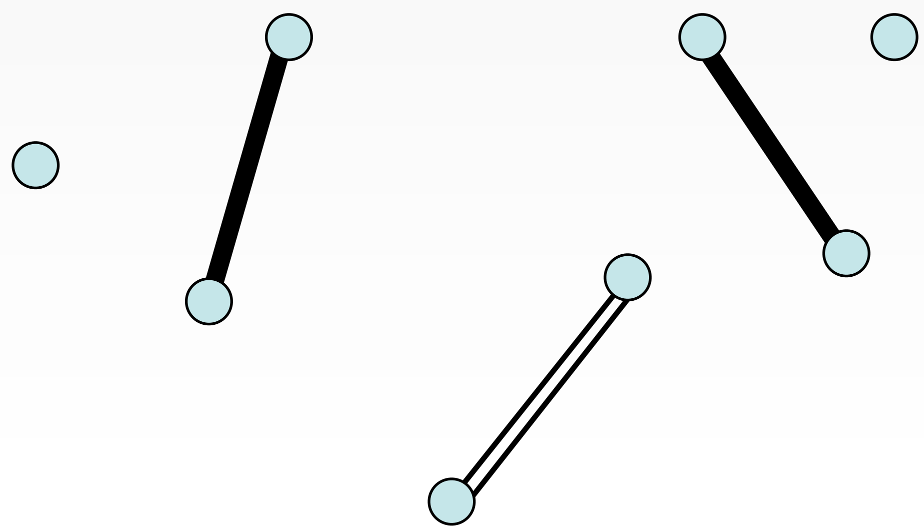
1. Take a random sample
2. Find a maximal matching on sample
3. Look at original graph



Algorithm

Given a graph:

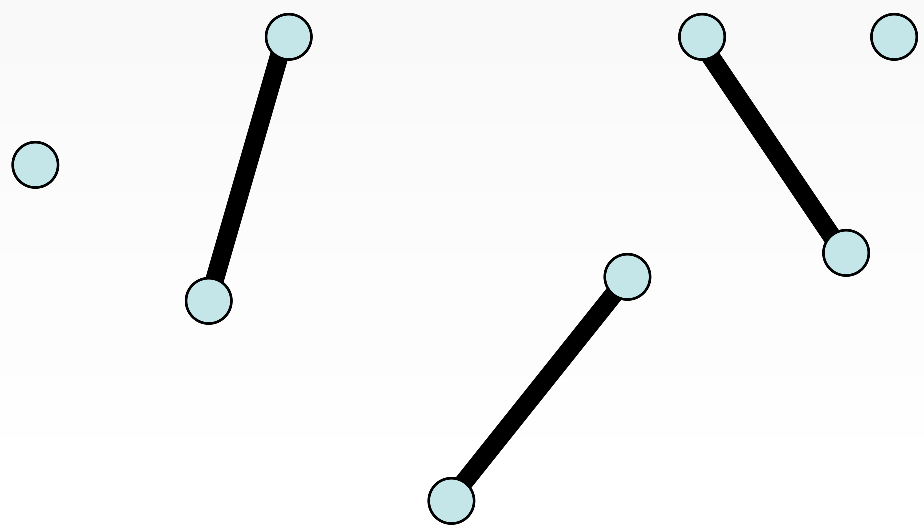
1. Take a random sample
2. Find a maximal matching on sample
3. Look at original graph, drop dead edges



Algorithm

Given a graph:

1. Take a random sample
2. Find a maximal matching on sample
3. Look at original graph, drop dead edges
4. Find matching on remaining edges



Analysis

Key Lemma:

- Suppose the sampling rate is $p = \frac{n^{1+c}}{m}$ for some $c > 0$.
- Then with high probability the number of edges remaining after the prune step is at most:

$$\frac{2n}{p} = \frac{2m}{n^c}$$

Analysis

Key Lemma:

- Suppose the sampling rate is $p = \frac{n^{1+c}}{m}$ for some $c > 0$.
- Then with high probability the number of edges remaining after the prune step is at most:

$$\frac{2n}{p} = \frac{2m}{n^c}$$

- Proof [Sketch]:
  - Suppose $I \subseteq V$ are unmatched after prune step
  - I must be an independent set in the sampled graph
  - If $|E[I]| > O(n/p)$ then it is an I.S. with probability at most e^{-n}
  - Union bound over all 2^n sets I.

Analysis

Key Lemma:

- Suppose the sampling rate is $p = \frac{n^{1+c}}{m}$ for some $c > 0$.
- Then with high probability the number of edges remaining after the prune step is at most:

$$\frac{2n}{p} = \frac{2m}{n^c}$$

Corollaries:

- Given n^{1+c} memory, algorithm requires $O(1)$ rounds
- Given $O(n \log n)$ memory, algorithm requires $O(\frac{\log n}{\log \log n})$ rounds.
- PRAM simulations: $\Theta(\log n)$ rounds

Greedy Algorithms

Max k-Cover:

- Given $U = \{u_1, u_2, \ldots, u_n\}$ and sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$
- Select $\mathcal{S}^* \subseteq \mathcal{S}$ with $|\mathcal{S}^*| = k$, that maximizes $\bigcup_{S \in \mathcal{S}^*} S$

Classical Algorithm:

- Greedy. Iteratively add largest set
- Remove all of the covered elements
- Guarantees a $1 - 1/e$ approximation to the optimal

Outline

Introduction

MapReduce

MR Algorithmics

- Connected Components
- Matchings
- Greedy Algorithms

Open Problems

Not so Greedy Algorithms

Not so Greedy Algorithm:

- Iteratively add a set within $(1 + \epsilon)$ of the largest
- Remove all of the covered elements

Not so Greedy Algorithms

Not so Greedy Algorithm:

- Iteratively add a set within $(1+\epsilon)$ of the largest
- Remove all of the covered elements

Correctness:

- Make almost as much progress as regular greedy algorithm
- Get an $1 - 1/e - O(\epsilon)$ approximation

Streaming Algorithms

Intuition:

- Look for large sets first, then progressively look for smaller sets.
- Natural algorithm:
  - Define a threshold: $v_i = \frac{n}{(1+\epsilon)^i}$
  - Stream through elements, add sets of value at least v_1
  - Stream through elements, add sets of value at least v_2
  - ...until completion

Streaming Algorithms

Intuition:

- Look for large sets first, then progressively look for smaller sets.
- Natural algorithm:
  - Define a threshold: $v_i = \frac{n}{(1+\epsilon)^i}$
  - Stream through elements, add sets of value at least v_1
  - Stream through elements, add sets of value at least v_2
  - ...until completion

Analysis:

- Every time make a decision within $(1+\epsilon)$ of OPT
- Need $O(\frac{\log \max |S|}{\epsilon})$ passes

Parallel Algorithms

Simulate the Streaming Algorithm

- Given a threshold, v_i, find a sequence of sets each covering at least v_i elements
- Similar to maximal matching:
  - Run the streaming algorithm on a sample
  - Remove the sets whose weight falls below the threshold
  - Repeat until no sets meet the threshold

Parallel Algorithms

Simulate the Streaming Algorithm

- Given a threshold, v_i, find a sequence of sets each covering at least v_i elements
- Similar to maximal matching:
  - Run the streaming algorithm on a sample
  - Remove the sets whose weight falls below the threshold
  - Repeat until no sets meet the threshold

Theorem [MKVV '12]: If each set is sampled with probability $p = km^{-\delta}$ each phase of the streaming algorithm can be simulated in $O(1/\delta)$ rounds

Sample&Prune

High-level primitive:

- Take a sample of the input
- Evaluate the function on the sample
- Prune the input
- Repeat

Sample&Prune

High-level primitive:

- Take a sample of the input
- Evaluate the function on the sample
- Prune the input
- Repeat

Much more general:

- General class of greedy algorithms
- (Sub)modular function maximization subject to set of constraints
  - matroid constraints, multiple knapsack constraints

Outline

Introduction

MapReduce

MR Algorithmics

- Connected Components
- Matchings
- Greedy Algorithms

Open Problems

Overview

MapReduce:

- Interleaves sequential and parallel computation
- Many algorithms “embarrassingly parallel” and don’t use the full power of intermediate steps.

Overview

MapReduce:

- Interleaves sequential and parallel computation
- Many algorithms “embarrassingly parallel” and don’t use the full power of intermediate steps.

What are the limits of the model:

- Can simulate PRAM algorithms. But are there bigger speedups to be had?
- No known lower bounds!
- Even a simpler model is interesting
  - e.g. assume data is partitioned and no replication as allowed

Classical Questions

Prefix Sum:

- Given an array `A[1...n]` compute `B[1...n]` where `B[i]=` $\sum_{j \leq i} A[j]$
- Relatively simple in MR (2 rounds suffices)

Classical Questions

Prefix Sum:

- Given an array `A[1...n]` compute `B[1...n]` where `B[i]=` $\sum_{j \leq i} A[j]$
- Relatively simple in MR (2 rounds suffices)

List Ranking:

- Given a permutation as a set of pointers to the next element: `P[i] = j`
- Find rank of each element (number of hops to reach it from `P[0]`)
- Easy in $O(\log n)$ rounds (simulate PRAM algorithm)
- Can you do it faster?

A Puzzle (essentially list ranking)

I give you an undirected graph on n nodes:

- As a list of edges
- With a promise that every vertex has degree exactly 2

You have:

- $n^{2/3}$ machines, each with $n^{2/3}$ memory

Your Mission:

- Tell me if the graph is connected in $o(\log n)$ rounds...
- ...or prove that it cannot be done

Thank You