Walking Randomly, Massively, and Efficiently

Krzysztof Onak

IBM Research

Joint work with Jakub Łącki (Google), Slobodan Mitrović (MIT), and Piotr Sankowski (University of Warsaw)
(Idealized) Past

Single machine directly accessing the entire data set
The Cloud

Setting: Data sets distributed across several machines

Why: full access by a small number of machines not feasible
The Cloud

Setting: Data sets distributed across several machines

Why: full access by a small number of machines not feasible

Massive data processing systems: MapReduce, Spark, Hadoop, Dryad, IBM Streams, Pregel, ...
The Cloud

Setting: Data sets distributed across several machines

Why: full access by a small number of machines not feasible

Develop algorithms that leverage the platform’s parallelism!

Massive data processing systems: MapReduce, Spark, Hadoop, Dryad, IBM Streams, Pregel, ...
Talk Plan

• The Massively Parallel Computation model

• Our Results

• Algorithms for Undirected Graphs
  + Lower Bounds
  + Applications to Property Testing

• Algorithms for Directed Graphs via Series of Transformations
Massively Parallel Computation
Model: Massively Parallel Computation (MPC)

\[ M \text{ machines} \quad \text{S space per machine} \]

Input: \( m \) edges from a graph on \( n \) vertices

- Initially: each machine receives \( \sim \frac{m}{M} \) edges
- Single round:
  1. Each machine performs computation
  2. Each machine sends and receives at most \( O(S) \) data
Model: Massively Parallel Computation (MPC)

$M$ machines \quad S \text{ space per machine}

Input: $m$ edges from a graph on $n$ vertices

- Initially: each machine receives $\sim m/M$ edges
Model: Massively Parallel Computation (MPC)

\[ M \text{ machines} \quad S \text{ space per machine} \]

Input: \( m \) edges from a graph on \( n \) vertices

- Initially: each machine receives \( \sim m/M \) edges
- Single round:
  1. Each machine performs computation
  2. Each machine sends and receives at most \( O(S) \) data
Model: Massively Parallel Computation (MPC)

- Introduced by Karloff, Suri, Vassilvitskii (2010) to model MapReduce due to Dean, Ghemawat (2004)

(from Dean, Ghemawat “MapReduce: Simplified Data Processing on Large Clusters”)
Model: Massively Parallel Computation (MPC)

• Introduced by Karloff, Suri, Vassilvitskii (2010) to model MapReduce due to Dean, Ghemawat (2004)

• Essential: space per machine $S = m^{\Omega(1)}$ (e.g., $S = \Omega(\sqrt{m})$)
Model: Massively Parallel Computation (MPC)

- Introduced by Karloff, Suri, Vassilvitskii (2010) to model MapReduce due to Dean, Ghemawat (2004)
- Essential: space per machine $S = m^{\Omega(1)}$ (e.g., $S = \Omega(\sqrt{m})$)
- Total space considerations:
  - [Beame 2009: Problem 27 at sublinear.info]
  - [Beame, Koutris, Suciu 2013]
  - [Andoni, Nikolov, Onak, Yaroslavtsev 2014]
  - Karloff et al. allow for $m^{1-\epsilon}$ machines with $m^{1-\epsilon}$ space
    $\Rightarrow$ near quadratic total space $m^{2-2\epsilon}$
Model: Massively Parallel Computation (MPC)

- Introduced by Karloff, Suri, Vassilvitskii (2010) to model MapReduce due to Dean, Ghemawat (2004)
- Essential: space per machine $S = m^{\Omega(1)}$ (e.g., $S = \Omega(\sqrt{m})$)
- Total space considerations:
  - Karloff et al. allow for $m^{1-\epsilon}$ machines with $m^{1-\epsilon}$ space $\Rightarrow$ near quadratic total space $m^{2-2\epsilon}$
  - A refined version asks for near-linear total space: $M \times S = m^{1+o(1)}$
Three Main Memory Regimes

- **Superlinear:** $S = n^{1+\Omega(1)}$
  - Many early papers [Karloff, Suri, Vassilvitskii 2010]
  - Round complexity: usually $O(1)$

- **Near-linear:** $S = \tilde{\Theta}(n)$
  - Not much was happening until 2017
  - Matchings, Vertex Cover, MIS in $O(\log \log n)$ rounds
  - Connectivity in $O(1)$ rounds
  - Very similar to the CONGESTED CLIQUE model

- **Sublinear:** $S = O(n^{\alpha})$ for $\alpha \in (0, 1)$
  - Most interesting for large sparse graphs
  - Results in this talk
  - Beating $O(\log n)$ becomes a challenge
Three Main Memory Regimes

- **Superlinear**: $S = n^{1+\Omega(1)}$
  - Many early papers [Karloff, Suri, Vassilvitskii 2010]
  - Round complexity: usually $O(1)$

- **Near-linear**: $S = \tilde{\Theta}(n)$
  - Not much was happening until 2017
  - Matchings, Vertex Cover, MIS in $O(\log \log n)$ rounds
  - Connectivity in $O(1)$ rounds
  - Very similar to the CONGESTED CLIQUE model

- **Sublinear**: $S = O(n^{\alpha})$ for $\alpha \in (0, 1)$
  - Most interesting for large sparse graphs
  - Results in this talk
  - Beating $O(\log n)$ becomes a challenge
Three Main Memory Regimes

- **Superlinear:** $S = n^{1+\Omega(1)}$
  - Many early papers [Karloff, Suri, Vassilvitskii 2010]
  - Round complexity: usually $O(1)$

- **Near-linear:** $S = \tilde{\Theta}(n)$
  - Not much was happening until 2017
  - Matchings, Vertex Cover, MIS in $O(\log \log n)$ rounds
  - Connectivity in $O(1)$ rounds
  - Very similar to the CONGESTED CLIQUE model

- **Sublinear:** $S = O(n^\alpha)$ for $\alpha \in (0, 1)$
  - Most interesting for large sparse graphs
  - Results in this talk
  - Beating $O(\log n)$ becomes a challenge
Main goal: minimize the number of rounds

• PRAM:
  • can usually be simulated in the same number of rounds
  • often $\Omega(\log n)$ rounds

![Diagram showing processors and data flow]

- Fewer parallel rounds than best PRAM algorithms?
  - $O(1)$ or $O(poly(\log \log N))$ rounds of MPC?
Main goal: minimize the number of rounds

• PRAM:
  • can usually be simulated in the same number of rounds
  • often $\Theta(\log n)$ rounds

• Fewer parallel rounds than best PRAM algorithms?

$O(1)$ or $O(\text{poly}(\log \log N))$ rounds of MPC?
Our Results
Random Walks

Why study random walks?

Useful primitive! Sample applications:

- PageRank and rating web pages
- optimal PRAM algorithms for connectivity
- partitioning graphs
- minimizing query complexity in property testing
- graph matchings in regular graphs
- generating random spanning trees
- volume estimation
- counting problems

Random walk on a graph:

At every step select a random outgoing edge

In general, the set of options could be weighted
Our Results

Setting: strongly sublinear space per machine, i.e., $O(n^\alpha)$ for $\alpha \in (0, 1)$

Generate a small number of length-$L$ random walks from every vertex

- undirected graphs: $O(\log L)$ rounds
- directed graphs: $O((\log \log n)^2 + \log^2 L)$ rounds

PageRank: $O((\log \log n)^2 + \log^2(1/\epsilon))$ rounds

- multiplicative approximation for all vertices
- $\epsilon = \text{teleportation probability}$
Undirected Graphs
Basic Challenges

**Trivial**: compute random walk of length $L$ in $O(L)$ rounds
Basic Challenges

**Trivial:** compute random walk of length $L$ in $O(L)$ rounds
Basic Challenges

**Trivial**: compute random walk of length $L$ in $O(L)$ rounds
Basic Challenges

**Trivial**: compute random walk of length $L$ in $O(L)$ rounds

Idea for more efficient algorithm:

- Random walks are memoryless
- Compute different sections and stitch them together?
- For $L$-step random walk, compute independently the first and second half of length $L/2$ via recursion?

Obstacles:

- We don’t know where the second $L/2$ steps start
- Compute many possible continuations?
- With many random walks, they could collide
Basic Challenges

Trivial: compute random walk of length $L$ in $O(L)$ rounds

Idea for more efficient algorithm:

• Random walks are memoryless
• Compute different sections and stitch them together?
• For $L$-step random walk, compute independently the first and second $L/2$ steps via recursion?

Obstacles:

• We don't know where the second $L/2$ steps start
• Compute many possible continuations?
• With many random walks, they could collide
Basic Challenges

**Trivial:** compute random walk of length $L$ in $O(L)$ rounds

Idea for more efficient algorithm:

- Random walks are memoryless
- Compute different sections and stitch them together?
- For $L$-step random walk, compute independently the first and second half of length $L/2$ via recursion?
Basic Challenges

Trivial: compute random walk of length $L$ in $O(L)$ rounds

Idea for more efficient algorithm:

- Random walks are memoryless
- Compute different sections and stitch them together?
- For $L$-step random walk, compute independently the first and second half of length $L/2$ via recursion?

Obstacles:

- We don’t know where the second $L/2$ steps start
- Compute many possible continuations?
Basic Challenges

Trivial: compute random walk of length $L$ in $O(L)$ rounds

Idea for more efficient algorithm:

- Random walks are memoryless
- Compute different sections and stitch them together?
- For $L$-step random walk, compute independently the first and second half of length $L/2$ via recursion?

Obstacles:

- We don’t know where the second $L/2$ steps start
- Compute many possible continuations?
- With many random walks, they could collide
Undirected Graphs

What we achieve: many random walks from each vertex
Undirected Graphs

What we achieve: many random walks from each vertex

How:

• Start from the stationary distribution: \( \frac{\text{deg}(v)}{2m} \) for vertex \( v \)
  • After any number of steps, the distribution will be the same

• Sample slightly more edges for consecutive steps to ensure that number of continuations is sufficient

• Roughly \( O(\text{deg}(v) \cdot \log n) \) random walks from vertex \( v \)

• Use \( O(\log L) \) rounds to combine edges into random walks
Is $O(\log L)$ Rounds Optimal?

- Space per machine: $S = n^\alpha$ for $\alpha \in (0, 1)$
- Problem: One or two cycles?

Best known algorithm: $O(\log n)$ rounds
Is $O(\log L)$ Rounds Optimal?

- Space per machine: $S = n^\alpha$ for $\alpha \in (0, 1)$
- Problem: One or two cycles?

Best known algorithm: $O(\log n)$ rounds

Good starting point for reductions:
- We show: if we can compute $O(\log^4 n)$–length random walks in $o(\log \log n)$ rounds, then this problem can be solved in $o(\log n)$ rounds
Is $O(\log L)$ Rounds Optimal?

- Space per machine: $S = n^\alpha$ for $\alpha \in (0, 1)$
- Problem: One or two cycles?

Best known algorithm: $O(\log n)$ rounds

Good starting point for reductions:
- We show: if we can compute $O(\log^4 n)$–length random walks in $o(\log \log n)$ rounds, then this problem can be solved in $o(\log n)$ rounds
- $\Omega(\log n)$ conditional lower bound for exact bipartiteness
Applications to Property Testing

Bipartiteness testing (similar to [Censor-Hillel, Fischer, Schwartzman, Vasudev 2016]):

- [Goldreich Ron 1999]: sampling $O(\sqrt{n})$ random walks from a random vertex is likely to detect an odd length cycle
- Can as well sample $O(1)$ random walks from all vertices

Testing if a graph is an expander:

- Expander: two random walks collide with probability close to $1/n$
- Far from expander: higher probability for a random starting vertex
- Tweak the proof of Czumaj and Sohler (2007) to distribute starting points of random walks over all vertices

Open question: Can this be done for testing clusterability?
Applications to Property Testing

Bipartiteness testing (similar to [Censor-Hillel, Fischer, Schwartzman, Vasudev 2016]):

- [Goldreich Ron 1999]: sampling $O(\sqrt{n})$ random walks from a random vertex is likely to detect an odd length cycle
- Can as well sample $O(1)$ random walks from all vertices

Testing if a graph is an expander:

- Classic property testing: [Goldrech, Ron 2000][Czumaj, Sohler 2007]…
- Expander: two random walks collide with probability close to $1/n$
- Far from expander: higher probability for a random starting vertex
- Tweak the proof of Czumaj and Sohler (2007) to distribute starting points of random walks over all vertices
Applications to Property Testing

Bipartiteness testing (similar to [Censor-Hillel, Fischer, Schwartzman, Vasudev 2016]):

- [Goldreich Ron 1999]: sampling $O(\sqrt{n})$ random walks from a random vertex is likely to detect an odd length cycle
- Can as well sample $O(1)$ random walks from all vertices

Testing if a graph is an expander:

- Expander: two random walks collide with probability close to $1/n$
- Far from expander: higher probability for a random starting vertex
- Tweak the proof of Czumaj and Sohler (2007) to distribute starting points of random walks over all vertices

Open question: Can this be done for testing clusterability?
How About Directed Graphs?

Difficulties:

- No explicit stationary distribution
- Values can be as low as $n^{-\Omega(n)}$
PageRank for Undirected Graphs
Definition of PageRank

PageRank: measure of importance of nodes in a graph

- **Stationary distribution**
- **Random walk:**
  - with probability $1 - \epsilon$, follow a random outgoing edge
  - with probability $\epsilon$, teleport to uniformly selected vertex in the entire graph
- $\epsilon =$ teleportation probability
Alternate Definition

This process gives the same distribution [Breyer 2002]

- Select a vertex $v$ uniformly at random
- Walk on the Markov chain until teleportation from some vertex $u$
- $u$ distributed according to PageRank
Algorithm for Undirected PageRank

Algorithm:

- Know how to generate random walks on the underlying undirected graph, starting point selected uniformly.
Algorithm for Undirected PageRank

Algorithm:
- Know how to generate random walks on the underlying undirected graph, starting point selected uniformly
- Toss a biased coin at every step to decide if teleportation occurs
- Distribution of vertices right before teleportation is PageRank

Important note: This works for directed graphs as long as someone gives us a collection of random walks with uniformly selected starting points
Algorithm for Undirected PageRank

Algorithm:

- Know how to generate random walks on the underlying undirected graph, starting point selected uniformly
- Toss a biased coin at every step to decide if teleportation occurs
- Distribution of vertices right before teleportation is PageRank
- Need at most $O(\epsilon^{-1} \log n)$ random walks from every vertex
- All random walks will teleport whp. after $O(\epsilon^{-1} \log n)$ steps

Important note: This works for directed graphs as long as someone gives us a collection of random walks with uniformly selected starting points
PageRank for Balanced Directed Graphs
c-Balanced Directed Graph

- Constant $c \in (0, 1)$

- For every vertex vertex $v$,
  \[
  \text{outdeg}(v) \geq c \cdot \text{indeg}(b)
  \]

- Random incident edge is directed in the correct direction with non-trivial probability
Transformation

$G$ is $c$-balanced graph

$P_G = \text{PageRank transition probability matrix for } G$

$\bar{G} = \text{undirected version of } G$

$P_{\bar{G}} = \text{PageRank transition probability matrix for } \bar{G}$
Transformation

$G$ is $c$-balanced graph

$P_G = \text{PageRank transition probability matrix for } G$

$\bar{G} = \text{undirected version of } G$

$P_{\bar{G}} = \text{PageRank transition probability matrix for } \bar{G}$

Sequence: $0 = \delta_0 < \delta_1 < \ldots < \delta_{k-1} < \delta_k = 1$.

Intermediate PageRank transition probability matrices:

$$P_i = \delta_i P_G + (1 - \delta_i) P_{\bar{G}}$$
Transformation

$G$ is $c$-balanced graph

$P_G = $ PageRank transition probability matrix for $G$

$\bar{G}$ = undirected version of $G$

$P_{\bar{G}} = $ PageRank transition probability matrix for $\bar{G}$

Sequence: $0 = \delta_0 < \delta_1 < \ldots < \delta_{k-1} < \delta_k = 1$.

Intermediate PageRank transition probability matrices:

$$P_i = \delta_i P_G + (1 - \delta_i) P_{\bar{G}}$$

How:

- Know how to compute stationary distribution for $P_0$
- Want to compute stationary distribution for $P_k$
- We show how to move from $P_i$ to $P_{i+1}$ for $\delta_{i+1} - \delta_i \approx \frac{1}{\log \log n}$
Transition from $P_i$ to $P_{i+1}$

1. Use stationary distribution for $P_i$ to generate random walks for $P_i$
Transition from $P_i$ to $P_{i+1}$

1. Use stationary distribution for $P_i$ to generate random walks for $P_i$

2. Use rejection sampling to adjust probabilities of random walks:
   - Every time we take a step in the “wrong” direction, reject the walk with small probability, so they come from $P_{i+1}$
Transition from $P_i$ to $P_{i+1}$

1. Use stationary distribution for $P_i$ to generate random walks for $P_i$

2. Use rejection sampling to adjust probabilities of random walks:
   - Every time we take a step in the “wrong” direction, reject the walk with small probability, so they come from $P_{i+1}$

3. Use the surviving random walks to compute PageRank for $P_{i+1}$
PageRank
for General Directed Graphs
Replacing Vertices with Paths

- Previous approach does not work for general graphs: would need a lot of samples at a vertex with few outgoing edges but lots of coming in.
Replacing Vertices with Paths

• Previous approach does not work for general graphs: would need a lot of samples at a vertex with few outgoing edges but lots of coming in

• Replace vertices $v$ with directed $O(\log n)$-paths
  • $i$-th edge: $\max n/2^i, \text{indeg}(v)$ copies

Correspondingly lower the teleportation probability

Transition from $\epsilon = 1/2$ to $\epsilon/\log n$ (again via series of transitions)
Replacing Vertices with Paths

• Previous approach does not work for general graphs: would need a lot of samples at a vertex with few outgoing edges but lots of coming in

• Replace vertices \( v \) with directed \( O(\log n) \)-paths
  • \( i \)-th edge: \( \max n/2^i, \text{indeg}(v) \) copies

• Correspondingly lower the teleportation probability

• Transition from \( \epsilon = 1/2 \) to \( \epsilon / \log n \)
  (again via series of transitions)
Generating Random Walks in Directed Graphs
Generating Random Walks in Directed Graphs

- $L =$ length of desired random walks
- Leverage the fact that we know the associated PageRank
- Set the teleportation probability to $1/L$
- Generate random walks from the PageRank Markov Chain
- Throw away those that teleported at least once
- A random walk “survives” with probability $\Omega(1)$
Open Questions
Open Questions

1. Are the “squares” needed? $O(\log L)$ rounds for directed random walks?
1. Are the “squares” needed? $O(\log L)$ rounds for directed random walks?

2. Testing clusterability?
Open Questions

1. Are the “squares” needed? $O(\log L)$ rounds for directed random walks?

2. Testing clusterability?

3. More general study of property testing guarantees in MPC?
Open Questions

1. Are the “squares” needed? $O(\log L)$ rounds for directed random walks?
2. Testing clusterability?
3. More general study of property testing guarantees in MPC?