Distributed Algorithms for Connectivity and MST in Large Graphs with Efficient Local Computation

Eric Ajieren, Khalid Hourani, William K. Moses Jr., Gopal Pandurangan

University of Houston

January 4, 2022

Table of Contents

Table of Contents

The k-Machine Model

- study distributed algorithms for large-scale graphs

The k-Machine Model

- study distributed algorithms for large-scale graphs
- focus on connectivity and MST

The k-Machine Model

- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model

The k-Machine Model

- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model
  - $k \geq 2$ machines jointly perform computations on input graph

The k-Machine Model

- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model
  - $k \geq 2$ machines jointly perform computations on input graph
  - input graph distributed uniformly at random to k machines

The k-Machine Model

- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model
  - $k \geq 2$ machines jointly perform computations on input graph
  - input graph distributed uniformly at random to k machines
    - n nodes, m edges

The k-Machine Model

- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model
  - $k \geq 2$ machines jointly perform computations on input graph
  - input graph distributed uniformly at random to k machines
    - n nodes, m edges
    - each node and its adjacency list is given to a random machine

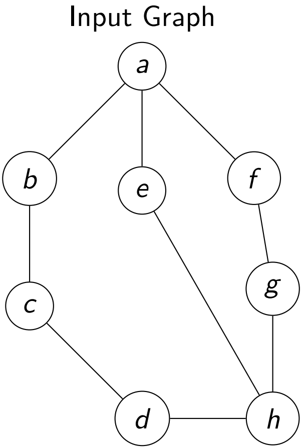
The k-Machine Model

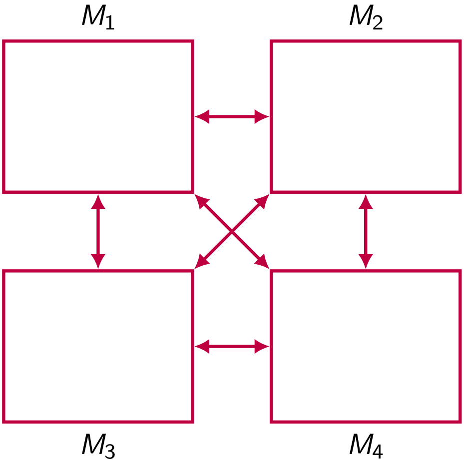
- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model
  - $k \geq 2$ machines jointly perform computations on input graph
  - input graph distributed uniformly at random to k machines
    - n nodes, m edges
    - each node and its adjacency list is given to a random machine
    - commonly called vertex centric model

The k-Machine Model

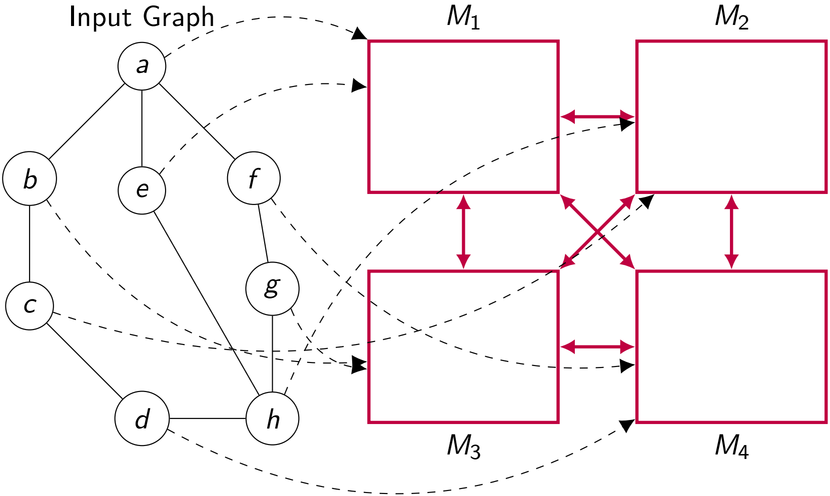
- study distributed algorithms for large-scale graphs
- focus on connectivity and MST
- in k-machine model
    - $k \geq 2$ machines jointly perform computations on input graph
    - input graph distributed uniformly at random to k machines
        - n nodes, m edges
        - each node and its adjacency list is given to a random machine
        - commonly called vertex centric model
    - assume $n \gg k$ (e.g. $k = \mathcal{O}(n^{\varepsilon})$)

k-Machine Model

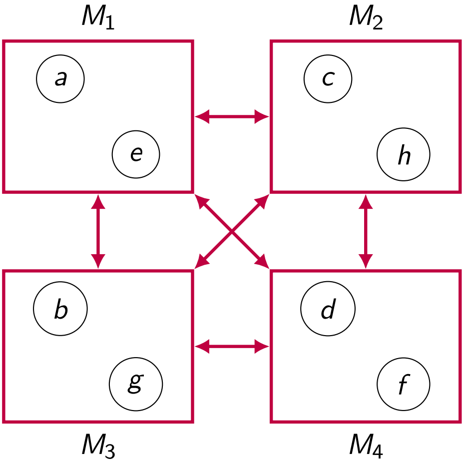




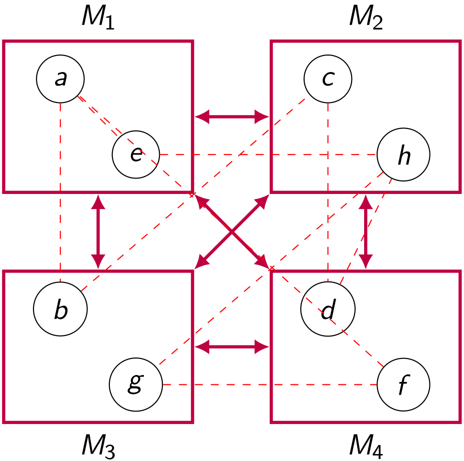
k-Machine Model



k-Machine Model



k-Machine Model



k-Machine Model

- typically, only consider communication cost

k-Machine Model

- typically, only consider communication cost
  - message complexity — number of communication rounds

k-Machine Model

- typically, only consider communication cost
  - message complexity — number of communication rounds
- traditionally done because network communication is much slower than local computation

k-Machine Model

- typically, only consider communication cost
  - message complexity — number of communication rounds
- traditionally done because network communication is much slower than local computation
  - less true as network speeds increase

k-Machine Model

- typically, only consider communication cost
  - message complexity — number of communication rounds
- traditionally done because network communication is much slower than local computation
  - less true as network speeds increase
  - for larger inputs, local computation becomes more significant

k-Machine Model

- typically, only consider communication cost
  - message complexity — number of communication rounds
- traditionally done because network communication is much slower than local computation
  - less true as network speeds increase
  - for larger inputs, local computation becomes more significant
- in practical implementations of k-machine model algorithms, often see suboptimal speedup

k-Machine Model

- typically, only consider communication cost
  - message complexity — number of communication rounds
- traditionally done because network communication is much slower than local computation
  - less true as network speeds increase
  - for larger inputs, local computation becomes more significant
- in practical implementations of k-machine model algorithms, often see suboptimal speedup
  - e.g. PageRank with message complexity $\mathcal{O}(n/k)$ but whose wall-clock time is approximately $n/k^{0.8}$

Augmenting the k-Machine Model with Local Computation

- we posit a new complexity measure

Augmenting the k-Machine Model with Local Computation

- we posit a new complexity measure
  - local computation cost — T_ℓ

Augmenting the k-Machine Model with Local Computation

- we posit a new complexity measure
  - local computation cost — T_ℓ
  - measures the worst-case local computation cost among k machines

Augmenting the k-Machine Model with Local Computation

- we posit a new complexity measure
  - local computation cost — T_ℓ
  - measures the worst-case local computation cost among k machines
- we refer to traditional communication complexity by T_c

Augmenting the k-Machine Model with Local Computation

- we posit a new complexity measure
  - local computation cost — T_ℓ
  - measures the worst-case local computation cost among k machines
- we refer to traditional communication complexity by T_c
- we analyze Connectivity and MST algorithms using this new complexity measure

Summary of Results

- we analyze several MST/CC algorithms under the new complexity measure

Summary of Results

- we analyze several MST/CC algorithms under the new complexity measure
- lower bounds for MST/CC

$$T_\ell = \Omega\left(\frac{m+n}{k} + \Delta + k\right)$$

$$T_c = \Omega\left(\frac{n}{k^2}\right)$$

Summary of Results

- we analyze several MST/CC algorithms under the new complexity measure
- lower bounds for MST/CC

$$T_\ell = \Omega\left(\frac{m+n}{k} + \Delta + k\right)$$

$$T_c = \Omega\left(\frac{n}{k^2}\right)$$

-

Algorithm	Round complexity	Local runtime
Flooding	$\tilde{O}\left(\frac{n}{k} + D\right)$	$\tilde{O}\left(\frac{m}{k} + \Delta + k\right)$
Filtering	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m}{k} + n\right)$
Borůvka-Style	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
Randomized CC	$\tilde{O}\left(\frac{n}{k^2}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$

Some Useful Lemmas

- as a byproduct of our analysis, we have two useful lemmas for analysis of k-Machine Model algorithms

Some Useful Lemmas

- as a byproduct of our analysis, we have two useful lemmas for analysis of k-Machine Model algorithms
  - the Node Distribution Lemma

Some Useful Lemmas

- as a byproduct of our analysis, we have two useful lemmas for analysis of k-Machine Model algorithms
  - the Node Distribution Lemma
  - the Mapping Lemma

Node Distribution Lemma

Lemma (Node Distribution Lemma)

Consider a graph G of nodes $v_1, v_2, \ldots, v_n$ with associated non-negative real-valued "weights" $w(v)$ for each node v. Given a uniform, random distribution of the n nodes to k machines, as in the k-machine model, then, with probability at least $1 - 1/n^a$ for any $a > 0$, the total weight of nodes at every machine is bounded above by

$$\mathcal{O}(T_{avg} + \log n \cdot w_{max})$$

where $T_{avg} = \frac{1}{k}\sum_{i=1}^{n} w(v_i)$ and $w_{max} = max\{w(v_i)\}$.

Mapping Lemma

Lemma (Mapping Lemma)

Let an n-node, m-edge graph G be partitioned among the k machines as $N = \{p_1, \ldots, p_k\}$*. Then with probability at least* $1 - 1/n^{\alpha}$*, where* $\alpha > 1$ *is an arbitrary fixed constant, the following bounds hold:*

Mapping Lemma

Lemma (Mapping Lemma)

Let an n-node, m-edge graph G be partitioned among the k machines as $N = \{p_1, \ldots, p_k\}$*. Then with probability at least* $1 - 1/n^{\alpha}$*, where* $\alpha > 1$ *is an arbitrary fixed constant, the following bounds hold:*

1. *The number of vertices mapped to any machine is* $\mathcal{O}(n/k)$*.*

Mapping Lemma

Lemma (Mapping Lemma)

Let an n-node, m-edge graph G be partitioned among the k machines as $N = \{p_1, \ldots, p_k\}$*. Then with probability at least* $1 - 1/n^{\alpha}$*, where* $\alpha > 1$ *is an arbitrary fixed constant, the following bounds hold:*

1. *The number of vertices mapped to any machine is* $\mathcal{O}(n/k)$*.*
2. *The number of edges mapped to any machine is* $\mathcal{O}(m/k + \Delta \log n)$*.*

Mapping Lemma

Lemma (Mapping Lemma)

Let an n-node, m-edge graph G be partitioned among the k machines as $N = \{p_1, \ldots, p_k\}$. Then with probability at least $1 - 1/n^{\alpha}$, where $\alpha > 1$ is an arbitrary fixed constant, the following bounds hold:

1. *The number of vertices mapped to any machine is $\mathcal{O}(n/k)$.*
2. *The number of edges mapped to any machine is $\mathcal{O}(m/k + \Delta \log n)$.*
3. *The number of edges mapped to any link of the network is $\mathcal{O}(m/k^2 + n/k)$.*

Table of Contents

Parallel Flooding

- the algorithm can be described from the perspective of a node

Parallel Flooding

- the algorithm can be described from the perspective of a node
  - each node chooses an ID uniformly at random in $[1, n^4]$

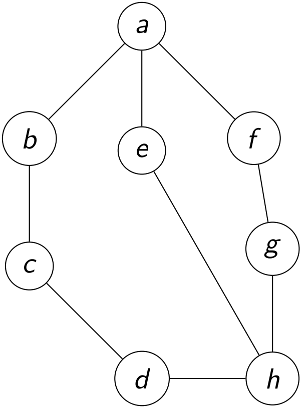
Parallel Flooding

- the algorithm can be described from the perspective of a node
  - each node chooses an ID uniformly at random in $[1, n^4]$
  - each node floods its ID

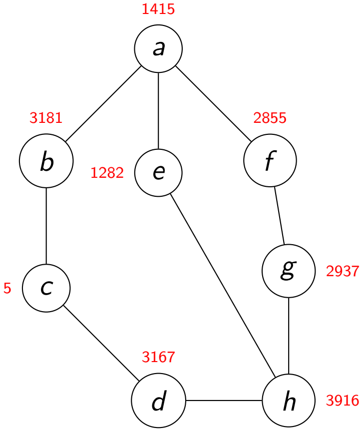
Parallel Flooding

- the algorithm can be described from the perspective of a node
  - each node chooses an ID uniformly at random in $[1, n^4]$
  - each node floods its ID
  - upon receiving a message (with an ID), if the new ID is greater than the current ID, the node updates its ID and floods

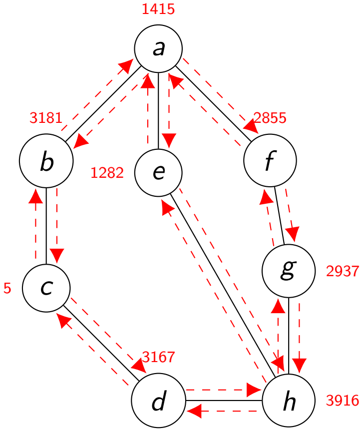
Parallel Flooding



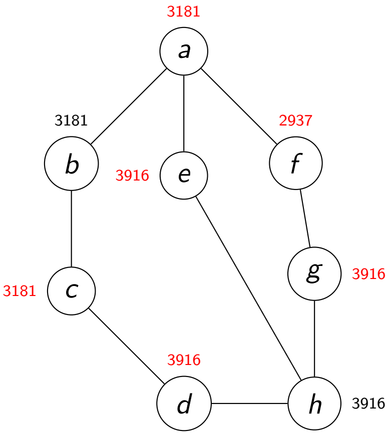
Parallel Flooding



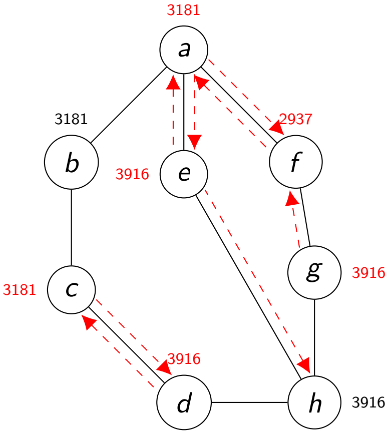
Parallel Flooding



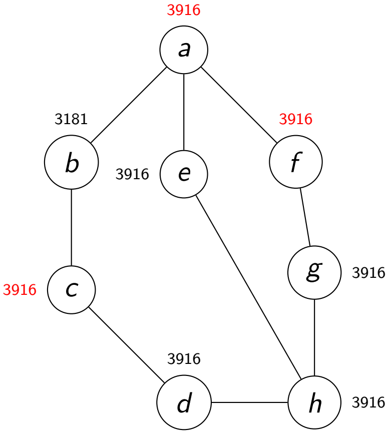
Parallel Flooding



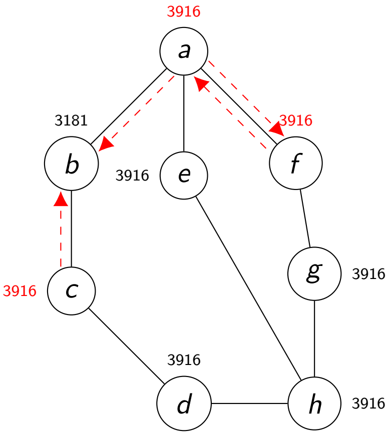
Parallel Flooding



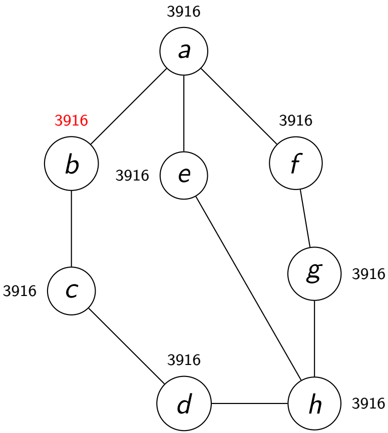
Parallel Flooding



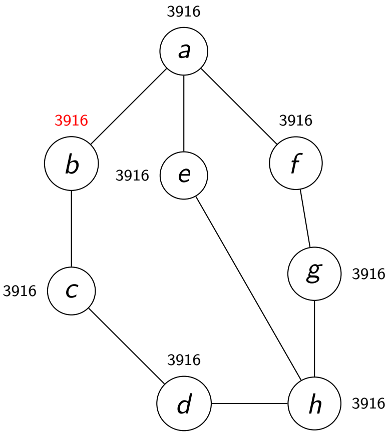
Parallel Flooding



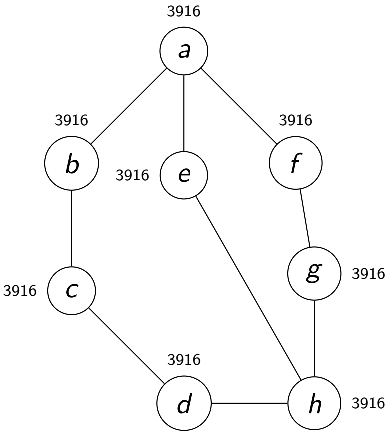
Parallel Flooding



Parallel Flooding



Parallel Flooding



Simulating in k-Machine Model

- in order to simulate this in the k-Machine Model, each machine M

Simulating in k-Machine Model

- in order to simulate this in the k-Machine Model, each machine M
  - maintains list of nodes on the machine in descending-order of ID

Simulating in k-Machine Model

- in order to simulate this in the k-Machine Model, each machine M
    - maintains list of nodes on the machine in descending-order of ID
    - iterates through list and simulates each node individually

Simulating in k-Machine Model

- in order to simulate this in the k-Machine Model, each machine M
    - maintains list of nodes on the machine in descending-order of ID
    - iterates through list and simulates each node individually
        - when u located on M sends a message to v on M', M sends the appropriate message to M'

Simulating in k-Machine Model

- in order to simulate this in the k-Machine Model, each machine M
  - maintains list of nodes on the machine in descending-order of ID
  - iterates through list and simulates each node individually
    - when u located on M sends a message to v on M', M sends the appropriate message to M'
  - after some $\mathcal{O}(n \log n/k + D)$ rounds, aggregate all IDs located on machine M and send them to machine M_1

Simulating in k-Machine Model

- in order to simulate this in the k-Machine Model, each machine M
  - maintains list of nodes on the machine in descending-order of ID
  - iterates through list and simulates each node individually
    - when u located on M sends a message to v on M', M sends the appropriate message to M'
  - after some $\mathcal{O}(n \log n/k + D)$ rounds, aggregate all IDs located on machine M and send them to machine M_1
- Finally, machine M_1 counts the number of distinct IDs (which is the number of connected components) and broadcasts it to all machines

Theorem

With high probability, the above algorithm correctly counts the number of connected components with

$$T_\ell = \mathcal{O}((m/k + \Delta \log n) \log n + k)$$
$$T_c = \mathcal{O}(n \log n/k + D)$$

Theorem

With high probability, the above algorithm correctly counts the number of connected components with

$$T_\ell = \mathcal{O}((m/k + \Delta \log n) \log n + k)$$
$$T_c = \mathcal{O}(n \log n/k + D)$$

Proof.

$\square$

Theorem

With high probability, the above algorithm correctly counts the number of connected components with

$$T_\ell = \mathcal{O}((m/k + \Delta \log n) \log n + k)$$
$$T_c = \mathcal{O}(n \log n/k + D)$$

Proof.

- with Chernoff Bounds, can show that a node v will update max-ID $\mathcal{O}(\log n)$ times with probability $1 - 1/n^a$

□

Theorem

With high probability, the above algorithm correctly counts the number of connected components with

$$T_\ell = \mathcal{O}((m/k + \Delta \log n) \log n + k)$$
$$T_c = \mathcal{O}(n \log n/k + D)$$

Proof.

- with Chernoff Bounds, can show that a node v will update max-ID $\mathcal{O}(\log n)$ times with probability $1 - 1/n^a$
- v will receive $\mathcal{O}(\deg(v) \log n)$ higher IDs

$\square$

Theorem

With high probability, the above algorithm correctly counts the number of connected components with

$$T_\ell = \mathcal{O}((m/k + \Delta \log n) \log n + k)$$
$$T_c = \mathcal{O}(n \log n / k + D)$$

Proof.

- with Chernoff Bounds, can show that a node v will update max-ID $\mathcal{O}(\log n)$ times with probability $1 - 1/n^a$
- v will receive $\mathcal{O}(\deg(v) \log n)$ higher IDs
- when max-ID is updated, v sends to $\deg(v)$ nodes, totaling $\mathcal{O}(\deg(v) \log n)$

□

Theorem

With high probability, the above algorithm correctly counts the number of connected components with

$$T_\ell = \mathcal{O}((m/k + \Delta \log n) \log n + k)$$
$$T_c = \mathcal{O}(n \log n/k + D)$$

Proof.

- with Chernoff Bounds, can show that a node v will update max-ID $\mathcal{O}(\log n)$ times with probability $1 - 1/n^a$
- v will receive $\mathcal{O}(\deg(v) \log n)$ higher IDs
- when max-ID is updated, v sends to $\deg(v)$ nodes, totaling $\mathcal{O}(\deg(v) \log n)$
- maximum runtime is therefore $\mathcal{O}(\Delta \log n)$.

$\square$

Proof.

□

Proof.

- apply node distribution lemma on the degree of nodes

$$\begin{aligned} T_\ell &= \mathcal{O}\left(\frac{1}{k}\left(\sum_{i=1}^{n}(d(v_i)\log n)\right) + \log n \cdot \Delta \log n\right) \\ &= \mathcal{O}\left(\left(\frac{m}{k} + \Delta \log n\right)\log n\right) \end{aligned}$$

□

Proof.

- apply node distribution lemma on the degree of nodes

$$T_\ell = \mathcal{O}\left(\frac{1}{k}\left(\sum_{i=1}^{n}(d(v_i)\log n)\right) + \log n \cdot \Delta \log n\right)$$
$$= \mathcal{O}\left(\left(\frac{m}{k} + \Delta \log n\right)\log n\right)$$

- in the worst case, a machine sends/receives messages from all other machines, totaling

$$\mathcal{O}\left(\left(\frac{m}{k} + \Delta \log n\right)\log n + k\right)$$

□

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ with high probability.

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ with high probability.

Proof.

$\square$

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ with high probability.

Proof.

- total number of broadcasts is $\mathcal{O}(B \log n/(kW) + D)$ where W is the bandwidth

$\square$

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ with high probability.

Proof.

- total number of broadcasts is $\mathcal{O}(B \log n/(kW) + D)$ where W is the bandwidth
- taking $W = \mathcal{O}(\log n)$

□

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ *with high probability.*

Proof.

- total number of broadcasts is $\mathcal{O}(B \log n/(kW) + D)$ where W is the bandwidth
- taking $W = \mathcal{O}(\log n)$
- by mapping lemma, $\mathcal{O}(\log n)$ broadcasts for a particular node with probability $1 - 1/n^2$

$\square$

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ with high probability.

Proof.

- total number of broadcasts is $\mathcal{O}(B \log n/(kW) + D)$ where W is the bandwidth
- taking $W = \mathcal{O}(\log n)$
- by mapping lemma, $\mathcal{O}(\log n)$ broadcasts for a particular node with probability $1 - 1/n^2$
- by union bound, $\mathcal{O}(\log n)$ broadcasts for all nodes with probability $1 - 1/n$

□

Lemma

The communication complexity of the algorithm is $\mathcal{O}(n \log n/k + D)$ with high probability.

Proof.

- total number of broadcasts is $\mathcal{O}(B \log n/(kW) + D)$ where W is the bandwidth
- taking $W = \mathcal{O}(\log n)$
- by mapping lemma, $\mathcal{O}(\log n)$ broadcasts for a particular node with probability $1 - 1/n^2$
- by union bound, $\mathcal{O}(\log n)$ broadcasts for all nodes with probability $1 - 1/n$
- thus, $\mathcal{O}(n \log n)$ broadcasts initiated with high probability

$\square$

Table of Contents

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's
  - each machine stores a disjoint-set (union-find) $\mathcal{M}$

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's
  - each machine stores a disjoint-set (union-find) $\mathcal{M}$
  - algorithm works in "phases"

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's
  - each machine stores a disjoint-set (union-find) $\mathcal{M}$
  - algorithm works in "phases"
  - in each phase:

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's
  - each machine stores a disjoint-set (union-find) $\mathcal{M}$
  - algorithm works in "phases"
  - in each phase:
    - an MOE is found for each fragment

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's
  - each machine stores a disjoint-set (union-find) $\mathcal{M}$
  - algorithm works in "phases"
  - in each phase:
    - an MOE is found for each fragment
    - fragments are merged along MOEs

Borůvka-Style Algorithm

- we present a deterministic algorithm similar to Borůvka's
- a quick review of Borůvka's
  - each machine stores a disjoint-set (union-find) $\mathcal{M}$
  - algorithm works in "phases"
  - in each phase:
    - an MOE is found for each fragment
    - fragments are merged along MOEs
  - when exactly one fragment remains, it forms an MST

The Algorithm

- rather than broadcasting MOEs, simulate unicast version (as in GHS)

The Algorithm

- rather than broadcasting MOEs, simulate unicast version (as in GHS)
- additionally, we filter to reduce the number of edges in each machine to $\mathcal{O}(n)$

The Algorithm

- rather than broadcasting MOEs, simulate unicast version (as in GHS)
- additionally, we filter to reduce the number of edges in each machine to $\mathcal{O}(n)$
  - machines create MSFs of their local subgraphs using Kruskal's

The Algorithm

- rather than broadcasting MOEs, simulate unicast version (as in GHS)
- additionally, we filter to reduce the number of edges in each machine to $\mathcal{O}(n)$
  - machines create MSFs of their local subgraphs using Kruskal's
  - "discard" edges that are not part of local MSF

The Algorithm

- rather than broadcasting MOEs, simulate unicast version (as in GHS)
- additionally, we filter to reduce the number of edges in each machine to $\mathcal{O}(n)$
  - machines create MSFs of their local subgraphs using Kruskal's
  - "discard" edges that are not part of local MSF
  - the remaining (at most $k(n-1)$ edges whp) are the only MST edges reamining

The Algorithm

- rather than broadcasting MOEs, simulate unicast version (as in GHS)
- additionally, we filter to reduce the number of edges in each machine to $\mathcal{O}(n)$
  - machines create MSFs of their local subgraphs using Kruskal's
  - "discard" edges that are not part of local MSF
  - the remaining (at most $k(n-1)$ edges whp) are the only MST edges reamining
- algorithm then continues on remaining subgraphs

The Algorithm

- for a fixed machine M and fragment f, M determines the Local Minimum Outgoing Edge (LOE) and broadcasts

The Algorithm

- for a fixed machine M and fragment f, M determines the Local Minimum Outgoing Edge (LOE) and broadcasts
- from the broadcast of the `LOE`s, each machine locally determines the global MOE for any fragment it contains

The Algorithm

- for a fixed machine M and fragment f, M determines the Local Minimum Outgoing Edge (LOE) and broadcasts
- from the broadcast of the `LOE`s, each machine locally determines the global MOE for any fragment it contains
- the machine then merges fragments

Merging

- Note that we cannot merge all fragments at once, as that takes time proportial to the length of the fragment chain

Merging

- Note that we cannot merge all fragments at once, as that takes time proportial to the length of the fragment chain
- use a technique similar to controlled GHS

Merging

- Note that we cannot merge all fragments at once, as that takes time proportial to the length of the fragment chain
- use a technique similar to controlled GHS
  - create rooted tree F — each node is a fragment and there is an edge between nodes if they share an MOE

Merging

- Note that we cannot merge all fragments at once, as that takes time proportial to the length of the fragment chain
- use a technique similar to controlled GHS
  - create rooted tree F — each node is a fragment and there is an edge between nodes if they share an MOE
  - construct maximal matching (using e.g. Cole-Vishkin)

Merging

- Note that we cannot merge all fragments at once, as that takes time proportial to the length of the fragment chain
- use a technique similar to controlled GHS
    - create rooted tree F — each node is a fragment and there is an edge between nodes if they share an MOE
    - construct maximal matching (using e.g. Cole-Vishkin)
    - merge all matched edges and any edge where exactly one endpoint is matched

T_ℓ and T_c

- $T_c = \tilde{O}(n/k)$

T_ℓ and T_c

- $T_c = \tilde{O}(n/k)$
- $T_\ell = \tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$

T_ℓ and T_c

- $T_c = \tilde{O}(n/k)$
- $T_\ell = \tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
  - filtering requires $\tilde{O}(m/k + \Delta)$

T_ℓ and T_c

- $T_c = \tilde{O}(n/k)$
- $T_\ell = \tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
  - filtering requires $\tilde{O}(m/k + \Delta)$
  - matching requires $\tilde{O}(n/k)$ rounds

T_ℓ and T_c

- $T_c = \tilde{O}(n/k)$
- $T_\ell = \tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
  - filtering requires $\tilde{O}(m/k + \Delta)$
  - matching requires $\tilde{O}(n/k)$ rounds
  - additional $\tilde{O}(k)$ from a machine needing to process messages from $k-1$ other machines

Table of Contents

Sketch-Based Randomized Algorithm

- we analyze the 2018 SPAA algorithm of Pandurangan, Robinson, and Scquizzato

Sketch-Based Randomized Algorithm

- we analyze the 2018 SPAA algorithm of Pandurangan, Robinson, and Scquizzato
- optimal up to polylog factors in terms of round complexity

Sketch-Based Randomized Algorithm

- we analyze the 2018 SPAA algorithm of Pandurangan, Robinson, and Scquizzato
- optimal up to polylog factors in terms of round complexity
  - but $\tilde{\mathcal{O}}(n^2)$ local computation complexity

Sketch-Based Randomized Algorithm

- we analyze the 2018 SPAA algorithm of Pandurangan, Robinson, and Scquizzato
- optimal up to polylog factors in terms of round complexity
    - but $\tilde{\mathcal{O}}(n^2)$ local computation complexity
- the algorithm is similar to Borůvka's algorithm, with the output being a labeling of the nodes, such that nodes in the same component have the same label

Sketch-Based Randomized Algorithm

- we analyze the 2018 SPAA algorithm of Pandurangan, Robinson, and Scquizzato
- optimal up to polylog factors in terms of round complexity
  - but $\tilde{O}(n^2)$ local computation complexity
- the algorithm is similar to Borůvka's algorithm, with the output being a labeling of the nodes, such that nodes in the same component have the same label
- relies on linear graph sketches to efficiently merge multiple components

Sketches

- sketches are found by forming a vector of length $\binom{n}{2}$ for each node v and and appropriate choice of an $\mathcal{O}\left(\log^2 n\right) \times \binom{n}{2}$ matrix. This requires $\tilde{\Omega}\left(n^2\right)$ local computation time.

Sketches

- sketches are found by forming a vector of length $\binom{n}{2}$ for each node v and and appropriate choice of an $\mathcal{O}\left(\log^2 n\right) \times \binom{n}{2}$ matrix. This requires $\tilde{\Omega}\left(n^2\right)$ local computation time.
- can be improved by using the 2015 sketches of King, Kutten, and Thorup

Sketches

- sketches are found by forming a vector of length $\binom{n}{2}$ for each node v and and appropriate choice of an $\mathcal{O}\left(\log^2 n\right) \times \binom{n}{2}$ matrix. This requires $\tilde{\Omega}\left(n^2\right)$ local computation time.
- can be improved by using the 2015 sketches of King, Kutten, and Thorup
- instead of storing vectors of length $\binom{n}{2}$, a leader machine broadcasts an odd hash function

Sketches

- sketches are found by forming a vector of length $\binom{n}{2}$ for each node v and and appropriate choice of an $\mathcal{O}(\log^2 n) \times \binom{n}{2}$ matrix. This requires $\tilde{\Omega}(n^2)$ local computation time.
- can be improved by using the 2015 sketches of King, Kutten, and Thorup
- instead of storing vectors of length $\binom{n}{2}$, a leader machine broadcasts an odd hash function
- each component uses this hash function to find an MOE

Sketches

- sketches are found by forming a vector of length $\binom{n}{2}$ for each node v and and appropriate choice of an $\mathcal{O}\left(\log^2 n\right) \times \binom{n}{2}$ matrix. This requires $\tilde{\Omega}\left(n^2\right)$ local computation time.
- can be improved by using the 2015 sketches of King, Kutten, and Thorup
- instead of storing vectors of length $\binom{n}{2}$, a leader machine broadcasts an odd hash function
- each component uses this hash function to find an MOE
- brings it to $T_\ell = \tilde{\mathcal{O}}((m+n)/k + \Delta + k)$

Sketches

- sketches are found by forming a vector of length $\binom{n}{2}$ for each node v and and appropriate choice of an $\mathcal{O}\left(\log^2 n\right) \times \binom{n}{2}$ matrix. This requires $\tilde{\Omega}\left(n^2\right)$ local computation time.
- can be improved by using the 2015 sketches of King, Kutten, and Thorup
- instead of storing vectors of length $\binom{n}{2}$, a leader machine broadcasts an odd hash function
- each component uses this hash function to find an MOE
- brings it to $T_\ell = \tilde{\mathcal{O}}((m+n)/k + \Delta + k)$
  - optimal, up to polylog factors!

Table of Contents

Conclusion

- Augmented the k-machine model with a metric to measure local computation

Algorithm	Round complexity	Local runtime
Flooding	$\tilde{O}\left(\frac{n}{k} + D\right)$	$\tilde{O}\left(\frac{m}{k} + \Delta + k\right)$
Filtering	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m}{k} + n\right)$
Borůvka-Style	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
Randomized CC	$\tilde{O}\left(\frac{n}{k^2}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$

Conclusion

- Augmented the k-machine model with a metric to measure local computation
  - Many "optimal" algorithms are only optimal with respect to T_c

Algorithm	Round complexity	Local runtime
Flooding	$\tilde{O}\left(\frac{n}{k} + D\right)$	$\tilde{O}\left(\frac{m}{k} + \Delta + k\right)$
Filtering	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m}{k} + n\right)$
Borůvka-Style	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
Randomized CC	$\tilde{O}\left(\frac{n}{k^2}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$

Conclusion

- Augmented the k-machine model with a metric to measure local computation
    - Many "optimal" algorithms are only optimal with respect to T_c
- Analyzed well-known MIS/CC algorithms with respect to T_ℓ and T_c

Algorithm	Round complexity	Local runtime
Flooding	$\tilde{O}\left(\frac{n}{k} + D\right)$	$\tilde{O}\left(\frac{m}{k} + \Delta + k\right)$
Filtering	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m}{k} + n\right)$
Borůvka-Style	$\tilde{O}\left(\frac{n}{k}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$
Randomized CC	$\tilde{O}\left(\frac{n}{k^2}\right)$	$\tilde{O}\left(\frac{m+n}{k} + \Delta + k\right)$

Open Problems

- Analyze other graph problems in this model using

Open Problems

- Analyze other graph problems in this model using
- Implementing algorithms and comparing wall-clock time with T_ℓ and T_c

Open Problems

- Analyze other graph problems in this model using
- Implementing algorithms and comparing wall-clock time with T_ℓ and T_c
  - e.g. are there constants α and β (for a given set of hardware) such that wall-clock time is $\alpha T_\ell + \beta T_c$?