Massively Parallel Graph Algorithms

Krzysztof Nowicki

knowicki@cs.uni.wroc.pl

Institute of Computer Science, University of Wrocław

Warsaw, 07 October 2020

Research environment

Places (and people)

Research environment

Places (and people)

- University of Wrocław (with Tomasz Jurdziński)

Research environment

Places (and people)

- University of Wrocław (with Tomasz Jurdziński)
- ETH Zurich (with Mohsen Ghaffari)

Research environment

Places (and people)

- University of Wrocław (with Tomasz Jurdziński)
- ETH Zurich (with Mohsen Ghaffari)
- IBM Research (with Krzysztof Onak)

Research environment

Places (and people)

- University of Wrocław (with Tomasz Jurdziński)
- ETH Zurich (with Mohsen Ghaffari)
- IBM Research (with Krzysztof Onak)

Financial support

- NCN grant no 2017/25/B/ST6/02010 (Tomasz' OPUS13 grant)
- NCN grant no 2019/32/T/ST6/00566 (my ETIUDA7 grant)
- FNP START scholarship (year 2020)

Models of computation

Modern Approach to Parallel Computing

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Initial take on MapReduce

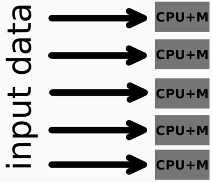
input data

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Initial take on MapReduce

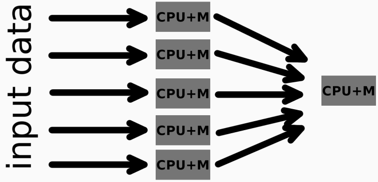


Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Initial take on MapReduce

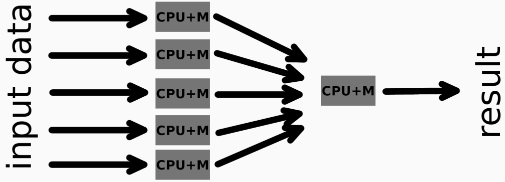


Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: Massively Parallel Computation (MPC) model

Initial take on MapReduce

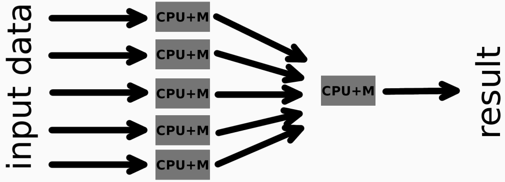


Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Initial take on MapReduce



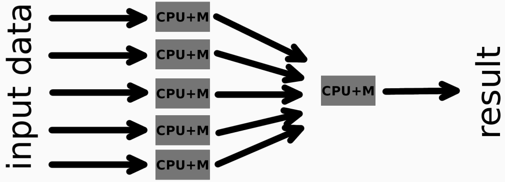
There are 2 kinds of problems:

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Initial take on MapReduce



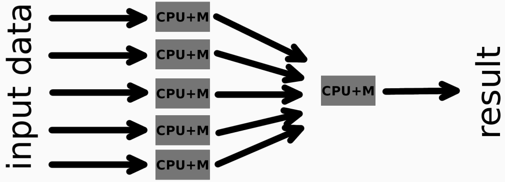
There are 2 kinds of problems:

- Easy: aggregative functions, e.g. sum, min, max, ...

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

Initial take on MapReduce



There are 2 kinds of problems:

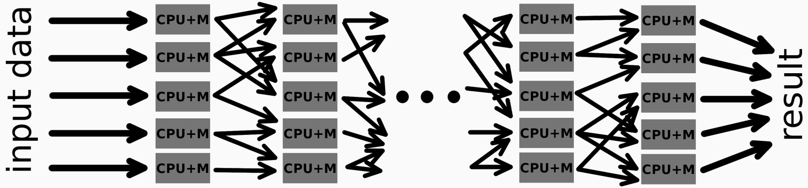
- Easy: aggregative functions, e.g. sum, min, max, ...
- Non trivial: some graph problems, e.g. Minimum Spanning Tree

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

MPC model

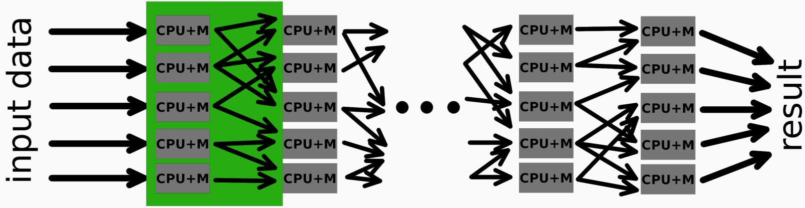


Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

MPC model



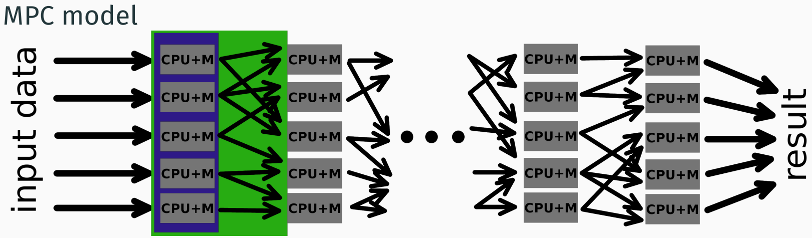
Computing in multiple rounds

- synchronous rounds

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model



Computing in multiple rounds

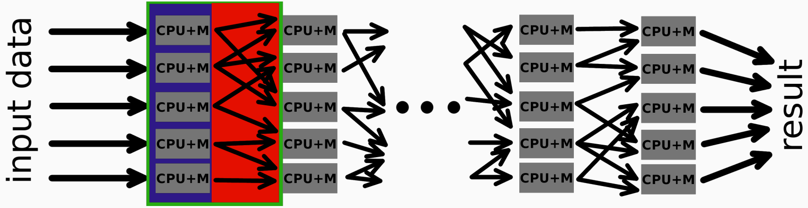
- synchronous rounds
- local computation

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

MPC model



Computing in multiple rounds

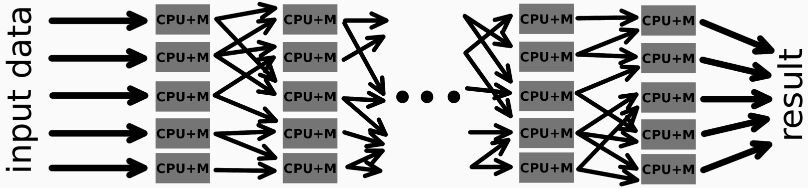
- synchronous rounds
- local computation
- communication

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

MPC model



Parameters

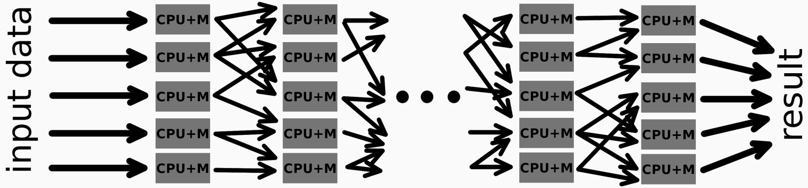
- each machine gets input of size S / has communication limit S

Modern Approach to Parallel Computing

Computational model

- industry: MapReduce, Spark, ...
- TCS community: **M**assively **P**arallel **C**omputation (**MPC**) model

MPC model



Parameters

- each machine gets input of size S / has communication limit S
- for input of size N, the number of machines is $\mathcal{O}(\frac{N}{S})$

Graph problems in the MPC model

Input / Output

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
    - single machine knows the whole result
    - each machine knows part of the result

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result
  - each machine knows part of the result

Three main variants

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result
  - each machine knows part of the result

Three main variants

- $S \in \Omega(n^{1+\epsilon})$, for a constant $\epsilon > 0$ (superlinear memory regime),

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result
  - each machine knows part of the result

Three main variants

- $S \in \Omega(n^{1+\epsilon})$, for a constant $\epsilon > 0$ (superlinear memory regime),
- $S \in \Theta(n)$ or $S \in \tilde{\Theta}(n)$ (linear memory regime),

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result
  - each machine knows part of the result

Three main variants

- $S \in \Omega(n^{1+\epsilon})$, for a constant $\epsilon > 0$ (superlinear memory regime),
- $S \in \Theta(n)$ or $S \in \tilde{\Theta}(n)$ (linear memory regime),
- $S \in \mathcal{O}(n^{1-\epsilon})$, for a constant $\epsilon > 0$ (sublinear memory regime).

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result
  - each machine knows part of the result

Three main variants

- $S \in \Omega(n^{1+\epsilon})$, for a constant $\epsilon > 0$ (superlinear memory regime) ,
- $S \in \Theta(n)$ or $S \in \tilde{\Theta}(n)$ (linear memory regime),
- $S \in \mathcal{O}(n^{1-\epsilon})$, for a constant $\epsilon > 0$ (sublinear memory regime).

Graph problems in the MPC model

Input / Output

- graph with n vertices and m edges
- initially: each machine knows arbitrary set of edges of size $\mathcal{O}(\frac{m}{S})$
- as a result:
  - single machine knows the whole result
  - each machine knows part of the result

Three main variants

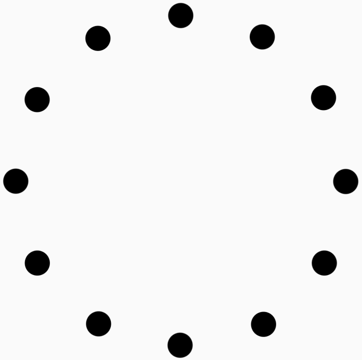
- $S \in \Omega(n^{1+\epsilon})$, for a constant $\epsilon > 0$ (superlinear memory regime) ,
- $S \in \Theta(n)$ or $S \in \tilde{\Theta}(n)$ (linear memory regime),
- $S \in \mathcal{O}(n^{1-\epsilon})$, for a constant $\epsilon > 0$ (sublinear memory regime).

Remark: $m = \#$ of edges, $n = \#$ of vertices, $\delta =$ min degree

Congested Clique Model

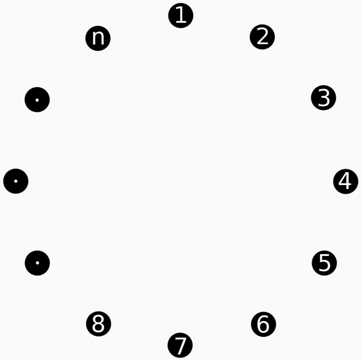
Congested Clique Model

- multi party communication



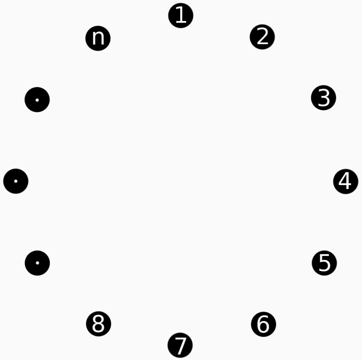
Congested Clique Model

- multi party communication
- n players with IDs in $[n]$



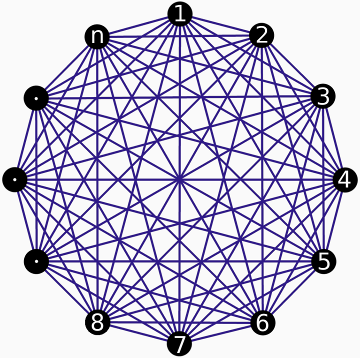
Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous



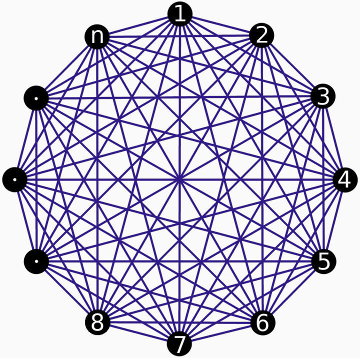
Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication



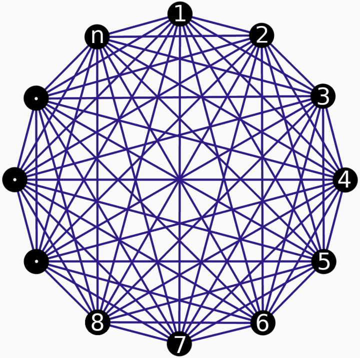
Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



Congested Clique Model

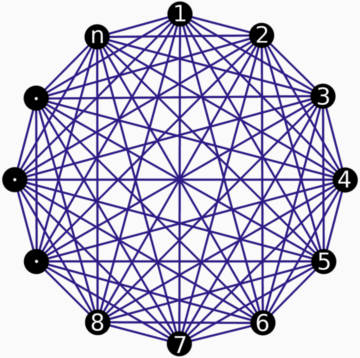
- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



Remarks

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages

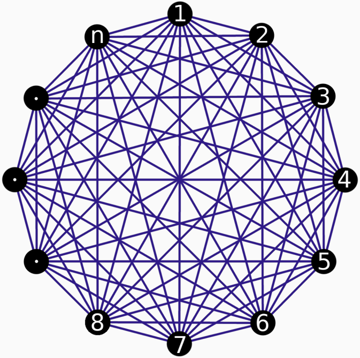


Remarks

- complexity: # of rounds

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages

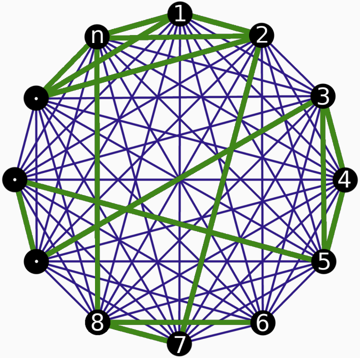


Remarks

- complexity: # of rounds
- $\Theta(n)$ different messages per player in a single round

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



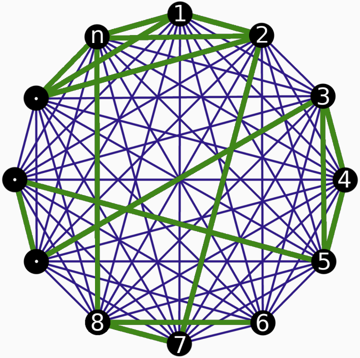
Remarks

- complexity: # of rounds
- $\Theta(n)$ different messages per player in a single round

Graph problems in the Congested Clique

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



Remarks

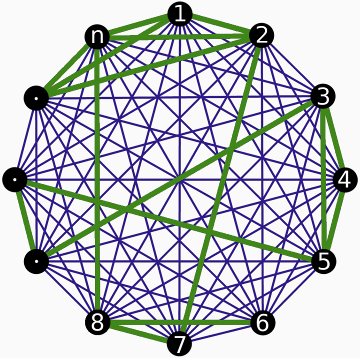
- complexity: # of rounds
- $\Theta(n)$ different messages per player in a single round

Graph problems in the Congested Clique

- each player represents single node of the input graph

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



Remarks

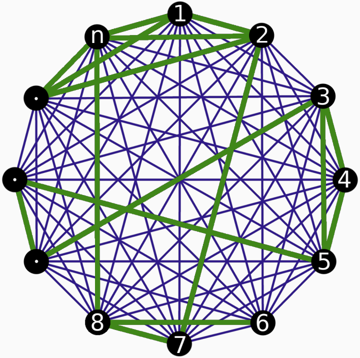
- complexity: # of rounds
- $\Theta(n)$ different messages per player in a single round

Graph problems in the Congested Clique

- each player represents single node of the input graph
- input: each player knows set of incident edges

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



Remarks

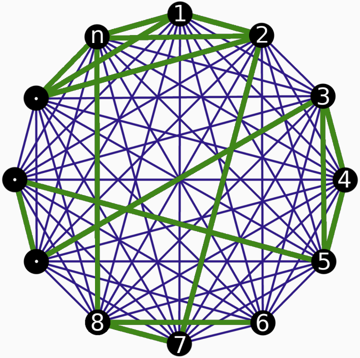
- complexity: # of rounds
- $\Theta(n)$ different messages per player in a single round

Graph problems in the Congested Clique

- each player represents single node of the input graph
- input: each player knows set of incident edges
- result: each player knows (part of) the output

Congested Clique Model

- multi party communication
- n players with IDs in $[n]$
- synchronous
- all-pair communication
- $\mathcal{O}(\log n)$ bit messages



Remarks

- complexity: # of rounds
- $\Theta(n)$ different messages per player in a single round

Almost equivalent to MPC with $\mathcal{O}(n)$ local memory.

Output of the community (recent FOCS / STOC / SODA)

Output of the community (recent FOCS / STOC / SODA)

*Dynamic Graph Algorithms with Batch Updates in the Massively Parallel Computation Model.(N., Onak; SODA'21)
*Walking Randomly, Massively, and Efficiently.(Łącki, Mitrović, Onak, Sankowski; STOC'20)
*Faster Algorithms for Edge Connectivity via Random 2-Out Contractions. (Ghaffari, N. Thorup, SODA'20)
*Parallel Batch-Dynamic Graphs: Constant Round Algorithms and Lower Bounds. (Durfee, Dhulipala, Kulkarni, Peng, Sawlani, Sun, SODA'20)
*Exponentially Faster Massively Parallel Maximal Matching. (Behnezhad, Hajiaghayi, Harris; FOCS'19)
*Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. (Ghaffari, Kuhn, Uitto; FOCS'19)
*Near-Optimal Massively Parallel Graph Connectivity. (Behnezhad, Dhulipala, Esfandiari, Łącki, Mirrokni; FOCS'19)
*Massively parallel approximation algorithms for edit distance and longest common subsequence(Hajiaghayi , Seddighin, Sun, SODA'19)
*Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation.(Ghaffari, Uitto; SODA'19)

Output of the community (not so recent FOCS / STOC / SODA)

*Parallel Graph Connectivity in Log Diameter Rounds.(Andoni, Song, Stein, Wang, Zhong; FOCS'18)
*Round compression for parallel matching algorithms.(Czumaj, Łącki, Mądry, Mitrović, Onak, Sankowski; STOC'18)
*MST in O(1) Rounds of Congested Clique.(Jurdzinski, N.; SODA'18)
*Efficient massively parallel methods for dynamic programming.(Im, Moseley, Sun; STOC'17)
*A New Framework for Distributed Submodular Maximization.(Barbosa, Ene, Nguyên, Ward; FOCS'16)
*Randomized Composable Core-sets for Distributed Submodular Maximization. (Mirrokni, Zadimoghaddam; STOC'15)
*Parallel Algorithms for Geometric Graph Problems.(Andoni, Nikolov, Onak, Yaroslavtsev, STOC'14)
*A Model of Computation for MapReduce. (Karloff, Suri, Vassilvitskii, SODA'10)

Output of the community (not so recent FOCS / STOC / SODA)

*Parallel Graph Connectivity in Log Diameter Rounds.(Andoni, Song, Stein, Wang, Zhong; FOCS'18)
*Round compression for parallel matching algorithms.(Czumaj, Łącki, Mądry, Mitrović, Onak, Sankowski; STOC'18)
*MST in O(1) Rounds of Congested Clique.(Jurdzinski, N.; SODA'18)
*Efficient massively parallel methods for dynamic programming.(Im, Moseley, Sun; STOC'17)
*A New Framework for Distributed Submodular Maximization.(Barbosa, Ene, Nguyên, Ward; FOCS'16)
*Randomized Composable Core-sets for Distributed Submodular Maximization. (Mirrokni, Zadimoghaddam; STOC'15)
*Parallel Algorithms for Geometric Graph Problems.(Andoni, Nikolov, Onak, Yaroslavtsev, STOC'14)
*A Model of Computation for MapReduce. (Karloff, Suri, Vassilvitskii, SODA'10)

Including other papers makes the list *significantly* longer.

My research

Three main topics.

Three main topics.

1. Minimum Spanning Tree problem

Three main topics.

1. Minimum Spanning Tree problem
2. Minimum Cut problem

Three main topics.

1. Minimum Spanning Tree problem
2. Minimum Cut problem
3. Dynamic Graph algorithms

My research

Minimum Spanning Tree problem

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source
1926	$\mathcal{O}(\log n)$	yes	Boruvka; Práce Mor. Přírodověd. Spol. V Brně

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source
1926	$\mathcal{O}(\log n)$	yes	Boruvka; Práce Mor. Přírodověd. Spol. V Brně
2003	$\mathcal{O}(\log \log n)$	yes	Lotker, Patt-Shamir, Pavlov, Peleg; SPAA'03

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source
1926	$\mathcal{O}(\log n)$	yes	Boruvka; Práce Mor. Přírodověd. Spol. V Brně
2003	$\mathcal{O}(\log \log n)$	yes	Lotker, Patt-Shamir, Pavlov, Peleg; SPAA'03
2015	$\mathcal{O}(\log \log \log n)$	no	Hegeman, Pandurangan, Pemmaraju, Sardeshmukh, Scquizzato; PODC'16

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source
1926	$\mathcal{O}(\log n)$	yes	Boruvka; Práce Mor. Přírodověd. Spol. V Brně
2003	$\mathcal{O}(\log\log n)$	yes	Lotker, Patt-Shamir, Pavlov, Peleg; SPAA'03
2015	$\mathcal{O}(\log\log\log n)$	no	Hegeman, Pandurangan, Pemmaraju, Sardeshmukh, Scquizzato; PODC'16
2016	$\mathcal{O}(\log^* n)$	no	Ghaffari, Parter; PODC'17

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source
1926	$\mathcal{O}(\log n)$	yes	Boruvka; Práce Mor. Přírodověd. Spol. V Brně
2003	$\mathcal{O}(\log \log n)$	yes	Lotker, Patt-Shamir, Pavlov, Peleg; SPAA'03
2015	$\mathcal{O}(\log \log \log n)$	no	Hegeman, Pandurangan, Pemmaraju, Sardeshmukh, Scquizzato; PODC'16
2016	$\mathcal{O}(\log^* n)$	no	Ghaffari, Parter; PODC'17
2018	$\mathcal{O}(1)$	no	Jurdziński, N.; SODA'18

MST -- a problem with a long history of results

year	round complexity	det.?	authors; source
1926	$\mathcal{O}(\log n)$	yes	Boruvka; Práce Mor. Přírodověd. Spol. V Brně
2003	$\mathcal{O}(\log \log n)$	yes	Lotker, Patt-Shamir, Pavlov, Peleg; SPAA'03
2015	$\mathcal{O}(\log \log \log n)$	no	Hegeman, Pandurangan, Pemmaraju, Sardeshmukh, Scquizzato; PODC'16
2016	$\mathcal{O}(\log^* n)$	no	Ghaffari, Parter; PODC'17
2018	$\mathcal{O}(1)$	no	Jurdziński, N.; SODA'18
2019	$\mathcal{O}(1)$	yes	N.; arxiv preprint

A sketch of a spanning forest algorithm

Hegeman et al. (PODC'15)

Instance of MST problem $\rightarrow$ several instances of Connected Components problem.

A sketch of a spanning forest algorithm

Hegeman et al. (PODC'15)

Instance of MST problem $\rightarrow$ several instances of Connected Components problem.

Spanning Forest algorithm

1. Select $\mathcal{O}(n)$ edges (in a certain way).
2. Compute connected components on selected edges.
3. Partition the component graph into edge disjoint subgraphs, constructed in a certain way.
4. Sparsify all subgraphs, in parallel.

A sketch of a spanning forest algorithm

Hegeman et al. (PODC'15)

Instance of MST problem $\rightarrow$ several instances of Connected Components problem.

Spanning Forest algorithm

1. Select $\mathcal{O}(n)$ edges (in a certain way).
2. Compute connected components on selected edges.
3. Partition the component graph into edge disjoint subgraphs, constructed in a certain way.
4. Sparsify all subgraphs, in parallel.

Sparsify = reduce the number of edges, while preserving connected components.

A sketch of a spanning forest algorithm

Hegeman et al. (PODC'15)
Instance of MST problem $\rightarrow$ several instances of Connected Components problem.

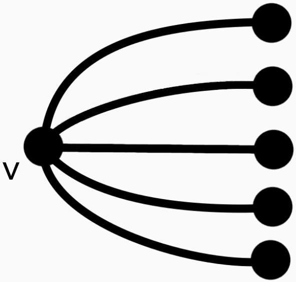
Spanning Forest algorithm

1. Select $\mathcal{O}(n)$ edges (in a certain way).
2. Compute connected components on selected edges.
3. Partition the component graph into edge disjoint subgraphs, constructed in a certain way.
4. Sparsify all subgraphs, in parallel.

Sparsify = reduce the number of edges, while preserving connected components.

A sketch of the spanning forest algorithm

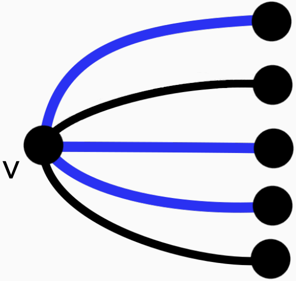
Edge selection



A sketch of the spanning forest algorithm

Edge selection

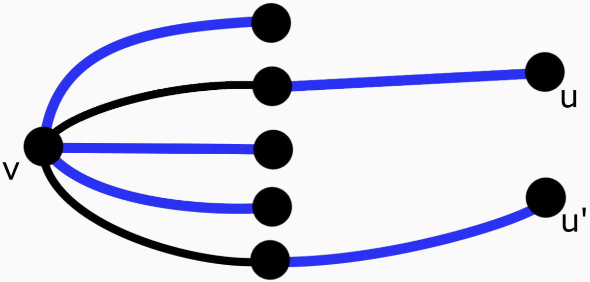
1. For each vertex mark an edge connecting it to the neighbour with max degree.



A sketch of the spanning forest algorithm

Edge selection

1. For each vertex mark an edge connecting it to the neighbour with max degree.

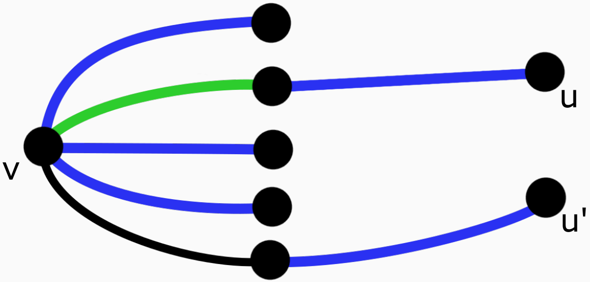


$(deg(u), id(u)) > (deg(v), id(v))$ $\qquad$ $(deg(u'), id(u')) > (deg(v), id(v))$

A sketch of the spanning forest algorithm

Edge selection

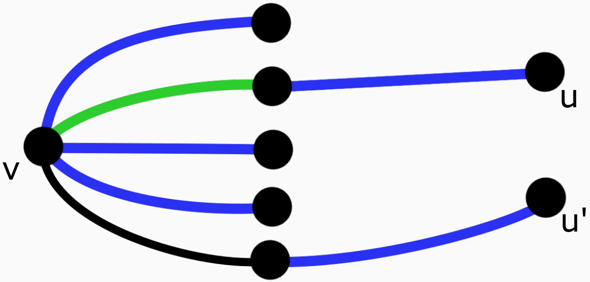
1. For each vertex mark an edge connecting it to the neighbour with max degree.
2. For each vertex mark an unmarked incident edge (if exists).



$(deg(u), id(u)) > (deg(v), id(v))$ $\quad$ $(deg(u'), id(u')) > (deg(v), id(v))$

Edge selection

1. For each vertex mark an edge connecting it to the neighbour with max degree.
2. For each vertex mark an unmarked incident edge (if exists).



$(deg(u), id(u)) > (deg(v), id(v)) \qquad (deg(u'), id(u')) > (deg(v), id(v))$

$\Rightarrow$ v in component of size at least $deg(v) + 1$.,

A sketch of the spanning forest algorithm

Spanning Forest algorithm

1. Select $\mathcal{O}(n)$ edges (in a certain way).
2. Compute connected components on selected edges.
3. Partition the component graph into edge disjoint subgraphs, *constructed in a certain way*.
4. Sparsify all subgraphs, in parallel.

A sketch of the spanning forest algorithm

Partition and sparsification

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:
A sparsification algorithm that reduces the number of edges to $\sqrt{mn}$, while preserving the edges of the minimum spanning forest.

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:
A sparsification algorithm that reduces the number of edges to $\sqrt{mn}$, while preserving the edges of the minimum spanning forest.
Korhonen's algorithm [B.A. at DISC 2016] adjusted to sparse graphs.

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:
A sparsification algorithm that reduces the number of edges to $\sqrt{mn}$, while preserving the edges of the minimum spanning forest. Korhonen's algorithm [B.A. at DISC 2016] adjusted to sparse graphs.

Partition:

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:
A sparsification algorithm that reduces the number of edges to $\sqrt{mn}$, while preserving the edges of the minimum spanning forest.
Korhonen's algorithm [B.A. at DISC 2016] adjusted to sparse graphs.

Partition:
$\mathcal{O}(\log n)$ edge disjoint subgraphs of the component graph

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:
A sparsification algorithm that reduces the number of edges to $\sqrt{mn}$, while preserving the edges of the minimum spanning forest.
Korhonen's algorithm [B.A. at DISC 2016] adjusted to sparse graphs.

Partition:
$\mathcal{O}(\log n)$ edge disjoint subgraphs of the component graph

$(V_1, E_1), (V_2, E_2), \ldots (V_{\mathcal{O}(\log n)}, E_{\mathcal{O}(\log n)})$

A sketch of the spanning forest algorithm

Partition and sparsification

Sparsification:
A sparsification algorithm that reduces the number of edges to $\sqrt{mn}$, while preserving the edges of the minimum spanning forest. Korhonen's algorithm [B.A. at DISC 2016] adjusted to sparse graphs.

Partition:
$\mathcal{O}(\log n)$ edge disjoint subgraphs of the component graph

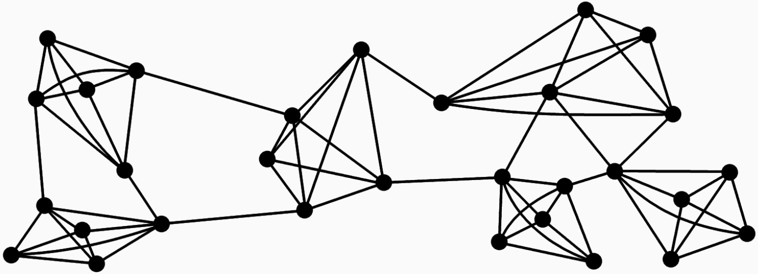
$(V_1, E_1), (V_2, E_2), \ldots (V_{\mathcal{O}(\log n)}, E_{\mathcal{O}(\log n)})$

s.t. $\sum_i \sqrt{|V_i| \cdot |E_i|} = \mathcal{O}(n)$.

My research

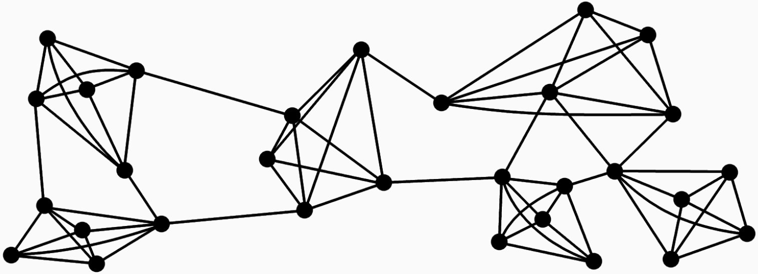
Minimum Cut problem

2-out contraction - algorithm for simple graphs



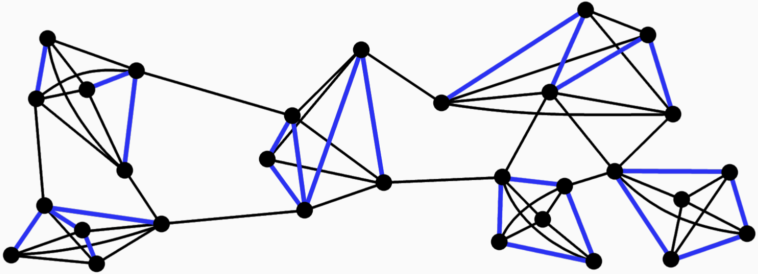
2-out contraction - algorithm for simple graphs

1. For each vertex mark 2 random incident edges.



2-out contraction - algorithm for simple graphs

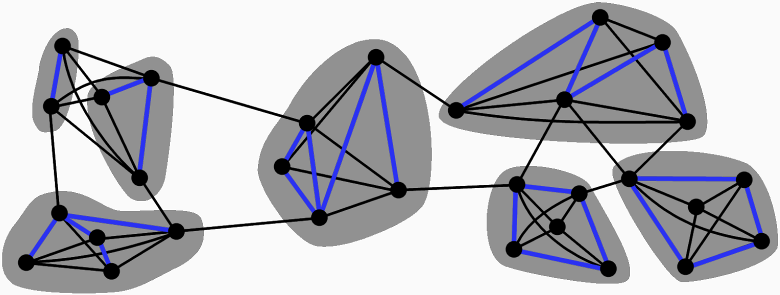
1. For each vertex mark 2 random incident edges.



2-out contractions (Ghaffari, N., Thorup; SODA'20)

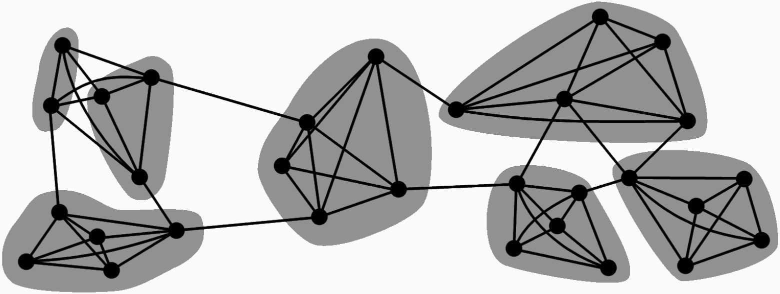
2-out contraction - algorithm for simple graphs

1. For each vertex mark 2 random incident edges.
2. Contract connected components spanned by marked edges.



2-out contractions (Ghaffari, N., Thorup; SODA'20)

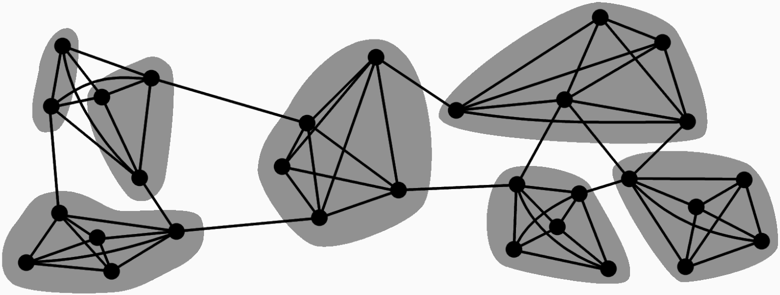
2-out contraction - properties (for simple graphs)



2-out contractions (Ghaffari, N., Thorup; SODA'20)

2-out contraction - properties (for simple graphs)

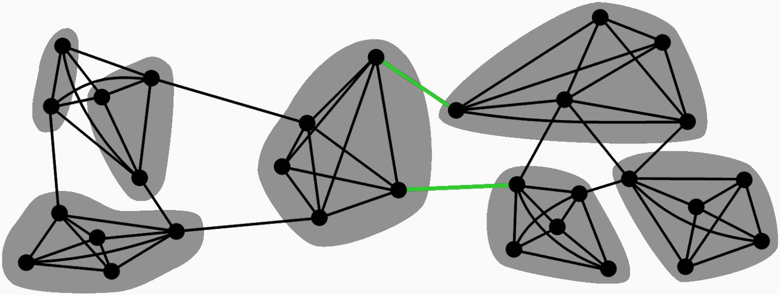
1. Number of vertices after contraction is $\mathcal{O}(n/\delta)$.



2-out contractions (Ghaffari, N., Thorup; SODA'20)

2-out contraction - properties (for simple graphs)

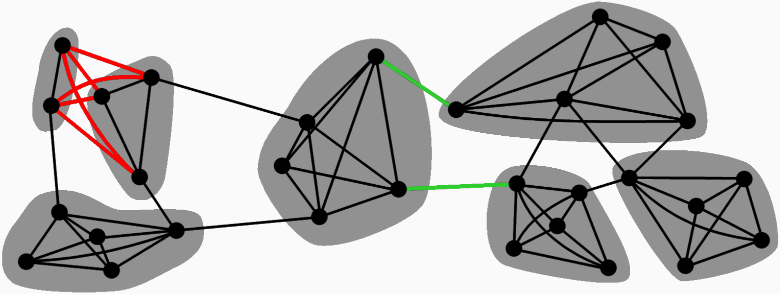
1. Number of vertices after contraction is $\mathcal{O}(n/\delta)$.
2. A fixed non-singleton min-cut is preserved with a constant probability.



2-out contractions (Ghaffari, N., Thorup; SODA'20)

2-out contraction - properties (for simple graphs)

1. Number of vertices after contraction is $\mathcal{O}(n/\delta)$.
2. A fixed non-singleton min-cut is preserved with a constant probability.



Algorithms based on 2-out

Missing pieces

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Improved Algorithms:

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Improved Algorithms:

1. Congested Clique / MPC: $\mathcal{O}(1)$ round algorithm.

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Improved Algorithms:

1. Congested Clique / MPC: $\mathcal{O}(1)$ round algorithm.
2. Sequential: $\min\left(\mathcal{O}(m\log n), \mathcal{O}(m + n\log^2 n)\right)$.

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Improved Algorithms:

1. Congested Clique / MPC: $\mathcal{O}(1)$ round algorithm.
2. Sequential: $\min\left(\mathcal{O}(m \log n), \mathcal{O}(m + n \log^2 n)\right)$.

impr. over Henzinger, Rao, Wang; SODA'17

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Improved Algorithms:

1. Congested Clique / MPC: $\mathcal{O}(1)$ round algorithm.
2. Sequential: $\min\left(\mathcal{O}(m \log n), \mathcal{O}(m + n \log^2 n)\right)$.
impr. over Henzinger, Rao, Wang; SODA'17
3. Distributed (Congest): $\tilde{O}(n^{0.8}D^{0.2} + n^{0.9})$.

Algorithms based on 2-out

Missing pieces

1. Reduce the number of edges (to $\mathcal{O}(n)$).
2. Modify alg. to preserve all small cuts with high probability.
3. Compute min-cut on smaller graph with different algorithm (for multigraphs).

Improved Algorithms:

1. Congested Clique / MPC: $\mathcal{O}(1)$ round algorithm.
2. Sequential: $\min\left(\mathcal{O}(m \log n), \mathcal{O}(m + n \log^2 n)\right)$.
impr. over Henzinger, Rao, Wang; SODA'17
3. Distributed (Congest): $\tilde{O}(n^{0.8} D^{0.2} + n^{0.9})$.
impr. over Daga, Henzinger, Nanongkai, Saranurak; STOC'19

MPC algorithms for weighted graphs (Ghaffari, N.; PODC'20)

MPC algorithms for weighted graphs (Ghaffari, N.; PODC'20)

Linear local memory

Adaptation of algorithm based on tree packings (Karger, STOC'96).

MPC algorithms for weighted graphs (Ghaffari, N.; PODC'20)

Linear local memory

Adaptation of algorithm based on tree packings (Karger, STOC'96).

Sublinear local memory

$(2+\varepsilon)$-approximation in $\mathcal{O}(\log n \log\log n)$ rounds, modification of contraction based algorithm for PRAM (Karger, SODA'94).

My research

Dynamic Graph algorithms

Motivation

In practice:

Motivation

In practice:

- solving problem repeatedly

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

- $G_1, G_2, \ldots, G_i$ is G_{i-1} with up to k modifications.

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

- $G_1, G_2, \ldots, G_i$ is G_{i-1} with up to k modifications.
- Goal: compute $S_1, S_2, \ldots,$ a sequence of solutions.

MPC: $S \in \Theta(n)$ vs $S \in \mathcal{O}(n^{1-\varepsilon})$

There is no proven separation between linear memory MPC and sublinear memory MPC. However, we know substantially better algorithms for $\mathcal{O}(n)$ memory regime.

	$S \in \mathcal{O}(n)$	$S \in \mathcal{O}(n^{1-\varepsilon})$
MST	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-EC	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
MIS	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$
MM	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

MPC: $S \in \Theta(n)$ vs $S \in \mathcal{O}(n^{1-\varepsilon})$

There is no proven separation between linear memory MPC and sublinear memory MPC. However, we know substantially better algorithms for $\mathcal{O}(n)$ memory regime.

	$S \in \mathcal{O}(n)$	$S \in \mathcal{O}(n^{1-\varepsilon})$
MST	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-EC	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
MIS	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$
MM	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

2-cycle Conjecture

Any algorithm that distinguish between single-cycle input and two-cycle input, with $\mathcal{O}(n^{1-\varepsilon})$ local memory, and $\mathrm{poly}(n)$ global memory requires $\Omega(\log n)$ rounds.

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

- $G_1, G_2, \ldots, G_i$ is G_{i-1} with up to k modifications.
- Goal: compute $S_1, S_2, \ldots$, a sequence of solutions.

Batch dynamic updates in MPC:

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

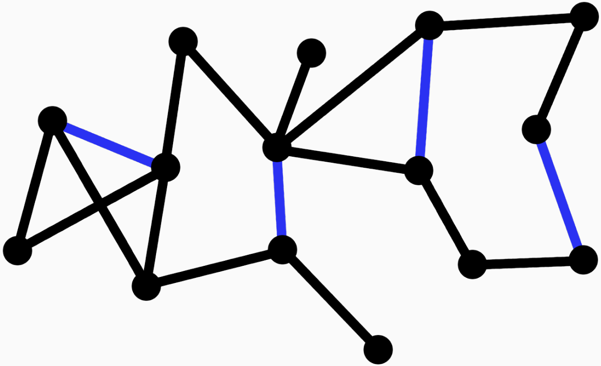
- $G_1, G_2, \ldots, G_i$ is G_{i-1} with up to k modifications.
- Goal: compute $S_1, S_2, \ldots,$ a sequence of solutions.

Batch dynamic updates in MPC:

- **Goal':** for batch-update on graph G_i compute update on S_i that transforms it into S_{i+1}

Example: Maximal Matching under batch updates

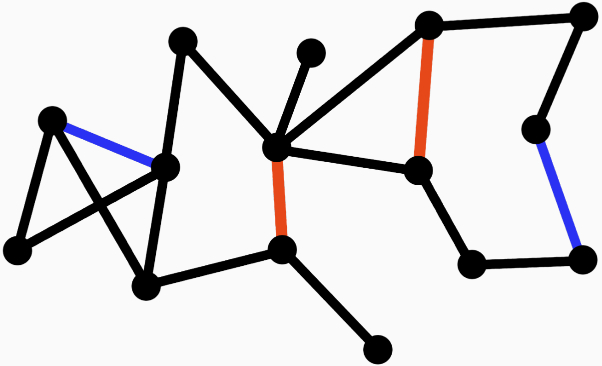
Input: Sequence of batch-updates, each adds / removes k edges.



Goal: Maintain a maximal matching; for each batch of updates to the graph, compute a batch of updates to the matching.

Example: Maximal Matching under batch updates

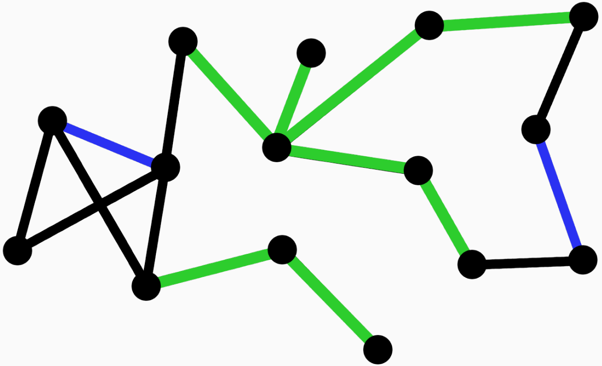
Input: Sequence of batch-updates, each adds / removes k edges.



Goal: Maintain a maximal matching; for each batch of updates to the graph, compute a batch of updates to the matching.

Example: Maximal Matching under batch updates

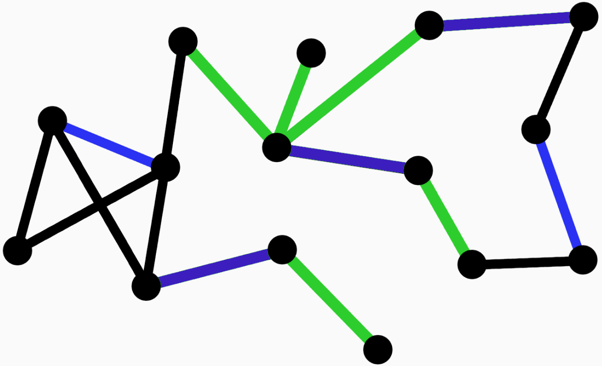
Input: Sequence of batch-updates, each adds / removes k edges.



Goal: Maintain a maximal matching; for each batch of updates to the graph, compute a batch of updates to the matching.

Example: Maximal Matching under batch updates

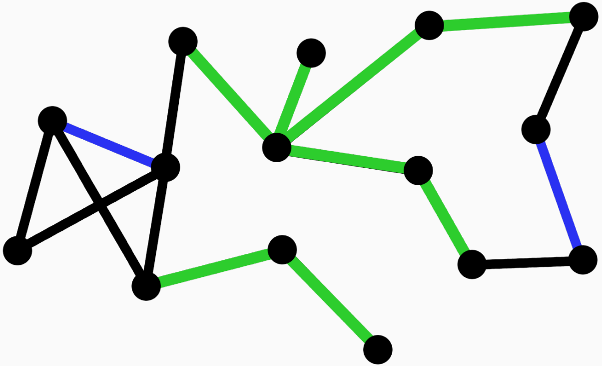
Input: Sequence of batch-updates, each adds / removes k edges.



Goal: Maintain a maximal matching; for each batch of updates to the graph, compute a batch of updates to the matching.

Example: Maximal Matching under batch updates

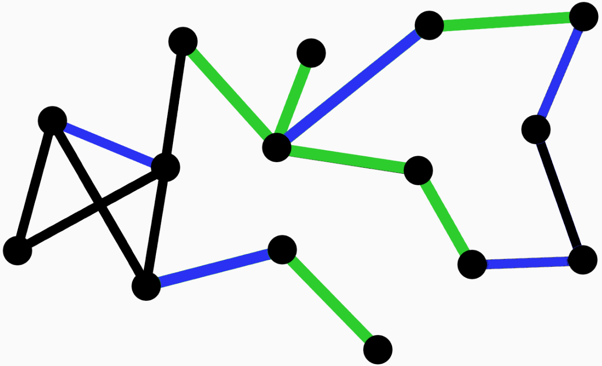
Input: Sequence of batch-updates, each adds / removes k edges.



Goal: Maintain a maximal matching; for each batch of updates to the graph, compute a batch of updates to the matching.

Example: Maximal Matching under batch updates

Input: Sequence of batch-updates, each adds / removes k edges.



Goal: Maintain a maximal matching; for each batch of updates to the graph, compute a batch of updates to the matching.

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

- $G_1, G_2, \ldots, G_i$ is G_{i-1} with up to k modifications.
- Goal: compute $S_1, S_2, \ldots,$ a sequence of solutions.

Batch dynamic updates in MPC:

- **Goal':** for batch-update on graph G_i compute update on S_i that transforms it into S_{i+1}
- **Meta-goal:** minimize round complexity of update and local memory S,

Motivation

In practice:

- solving problem repeatedly
- slightly changing data set

More formally

- $G_1, G_2, \ldots, G_i$ is G_{i-1} with up to k modifications.
- Goal: compute $S_1, S_2, \ldots,$ a sequence of solutions.

Batch dynamic updates in MPC:

- **Goal':** for batch-update on graph G_i compute update on S_i that transforms it into S_{i+1}
- **Meta-goal:** minimize round complexity of update and local memory S, while maximizing batch size k.

Dynamic Graph Algorithms with Batch Updates in the MPC (N., Onak; SODA'21)

Our results (for $k \in \mathcal{O}(S)$)

Problem	upd. compl.	ini. compl.
Minimum Spanning Tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-Edge Connected Components	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Maximal Matching	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

Dynamic Graph Algorithms with Batch Updates in the MPC (N., Onak; SODA'21)

Our results (for $k \in \mathcal{O}(S)$)

Problem	upd. compl.	ini. compl.
Minimum Spanning Tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-Edge Connected Components	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Maximal Matching	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

Our building blocks

1. Top tree implementation (for 1 and 2).
2. Additional observations (for 1 and 3).

Dynamic Graph Algorithms with Batch Updates in the MPC (N., Onak; SODA'21)

Our results (for $k \in \mathcal{O}(S)$)

Problem	upd. compl.	ini. compl.
Minimum Spanning Tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-Edge Connected Components	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Maximal Matching	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

Our building blocks

1. Top tree implementation (for 1 and 2).
2. Additional observations (for 1 and 3).

MST: reduce the problem to static version, with $\mathcal{O}(k)$ vertices

Dynamic Graph Algorithms with Batch Updates in the MPC (N., Onak; SODA'21)

Our results (for $k \in \mathcal{O}(S)$)

Problem	upd. compl.	ini. compl.
Minimum Spanning Tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-Edge Connected Components	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Maximal Matching	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

Our building blocks

1. Top tree implementation (for 1 and 2).
2. Additional observations (for 1 and 3).

MST: reduce the problem to static version, with $\mathcal{O}(k)$ vertices
MM: static version of the problem with vertex cover of size $\mathcal{O}(k)$

Dynamic Graph Algorithms with Batch Updates in the MPC (N., Onak; SODA'21)

Our results (for $k \in \mathcal{O}(S)$)

Problem	upd. compl.	ini. compl.
Minimum Spanning Tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-Edge Connected Components	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Maximal Matching	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

Our building blocks

1. Top tree implementation (for 1 and 2).
2. Additional observations (for 1 and 3).

MST: reduce the problem to static version, with $\mathcal{O}(k)$ vertices
MM: static version of the problem with vertex cover of size $\mathcal{O}(k)$

Better-than-static algorithms for $k \in \Theta(S^{1+\varepsilon'})$?

Dynamic Graph Algorithms with Batch Updates in the MPC (N., Onak; SODA'21)

Our results (for $k \in \mathcal{O}(S)$)

Problem	upd. compl.	ini. compl.
Minimum Spanning Tree	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
2-Edge Connected Components	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Maximal Matching	$\mathcal{O}(\log \log n)$	$\tilde{\mathcal{O}}(\sqrt{\log n})$

Our building blocks

1. Top tree implementation (for 1 and 2).
2. Additional observations (for 1 and 3).

MST: reduce the problem to static version, with $\mathcal{O}(k)$ vertices
MM: static version of the problem with vertex cover of size $\mathcal{O}(k)$

Better-than-static algorithms for $k \in \Theta(S^{1+\varepsilon'})$? Unlikely.

Thanks!

Thanks!

For the details:

- email me: knowicki@cs.uni.wroc.pl
- ask me in person