Batch Dynamic Algorithms

Guy Blelloch
Carnegie Mellon University

Joint work with Umut Acar, **Daniel Anderson**, Laxman Dhulipala, Tom Tseng, Kanat Tangwongsan, Sam Westrick

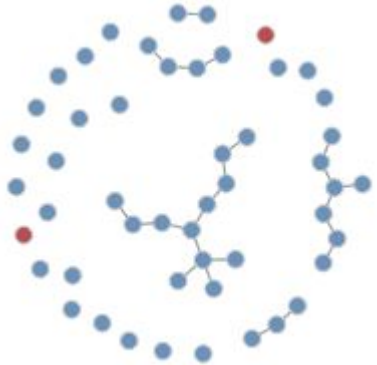
Dynamic Algorithms

- Efficiently support “non-trivial” queries and updates on a structure, e.g.,
  - check if an edge is in MST while inserting and deleting edges
  - check if a point is on the convex hull while inserting and deleting points
- Studied for 40+ years

  Many of you have worked on such algorithms
- Focus has been on 1 update at a time (e.g. inserting or deleting one edge)

Batch Dynamic Algorithms

- Support batches of operations (e.g. insert a batch of k edges)
- Goals:
  a. **Process batches in parallel** to improve performance
  b. Possibly also **reduce the cost relative to one at a time**
- Motivation 1: “big data” with rapid update rates
  a. The web graph: new and removed pages as crawled
  b. Social networks: friending/unfriending
  c. Time series of data (e.g. IP connections, points of sale)
- Motivation 2: substep in a parallel algorithm



Types of batches

1. **Incremental** : just insertions
2. **Fully dynamic** : insertions and deletions
3. **Sliding window**: insertions, and deletions of k oldest
4. ***Mixed**: sequence of interleaved updates and queries
5. **Kinetic**: moving data, batch processing of events

Preliminaries: Model

We use the “work-span” model.

Work is the total number of instructions
Span (also depth or critical path) is the longest chain of dependences

Several variants:

1. Binary vs. multi-way forking
2. Memory operations (test and set, CAS, ...)

Important property: bounds are **robust** across most models in terms of work, and at most O(log Work) different in span. Also true relative to the PRAM variants.

Our Work (1):

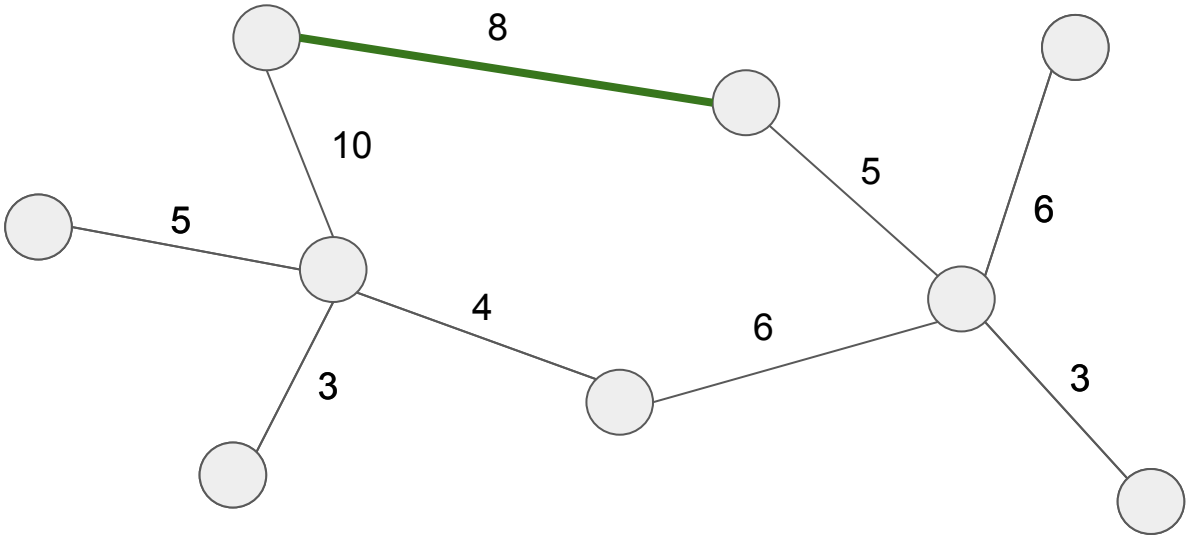
- **Batch-Parallel Euler Tour Trees** [Tseng, Dhulipala, B., 2019]
  k links or cuts on forest with n nodes:
  O(k log(n/k)) work, O(log n) span
  Note: for large k better than sequential
- **Parallel Batch-Dynamic Graph Connectivity** [Acar, Anderson, B., Dhuli., 2019]
  Insertions and deletions with average batch size k:
  O(log n log (n/k)) amortized work per update, polylog span per batch
- **Parallel Batch-Dynamic Trees via Change Propagation**
  [Acar, Anderson, B., Dhulipala, Westrick, 2020]
  Same bounds as Euler Tour Trees. Supports path queries

Our Work (2): **will talk about these today**

- **Work-Efficient Batch-Incremental Minimum Spanning Trees with Applications to the Sliding-Window Model** [Anderson, B., Tangwongsan, 2020]
  k inserts into MST of size n: O(k log(n/k)) work, polylog span
  Sliding window connectivity, cycle freeness, bipartiteness in same bound
- **Parallel Minimum Cuts in $O(m \log^2 n)$ work and Low Depth**
  [Anderson, B., 2021]
  k mixed updates and queries on a tree of size n:
  O(k log n) work and polylog span.

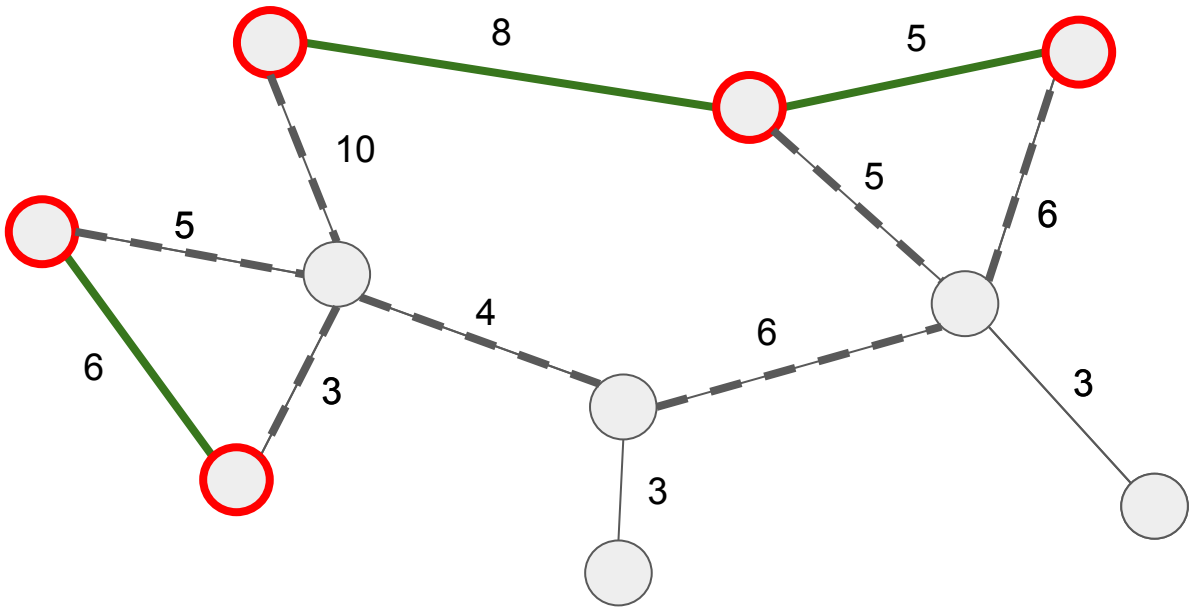
Parallel Batch-Incremental MSTs

Incremental MST (Single Update)

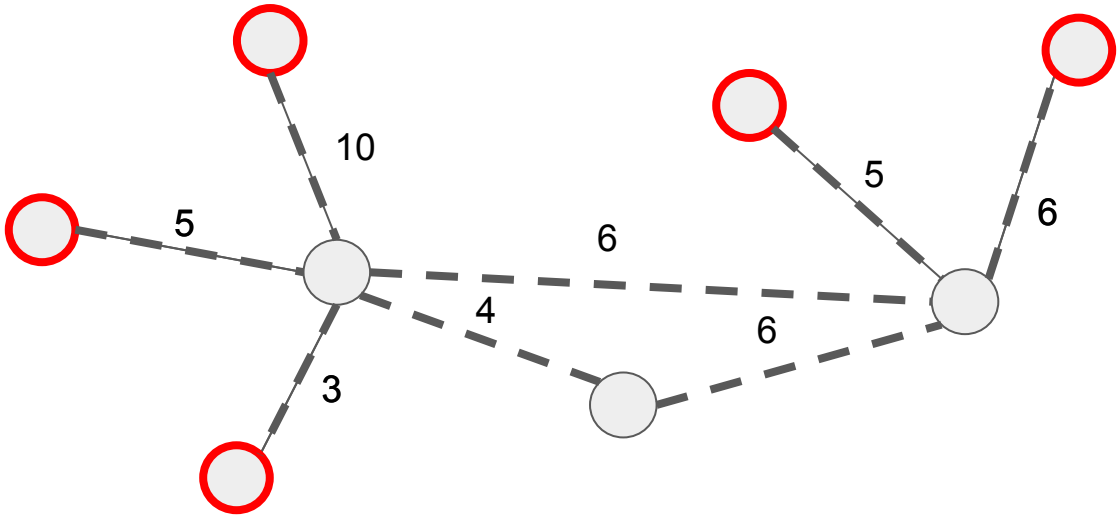


Link-Cut Trees [Sleator and Tarjan]
O(log n) time

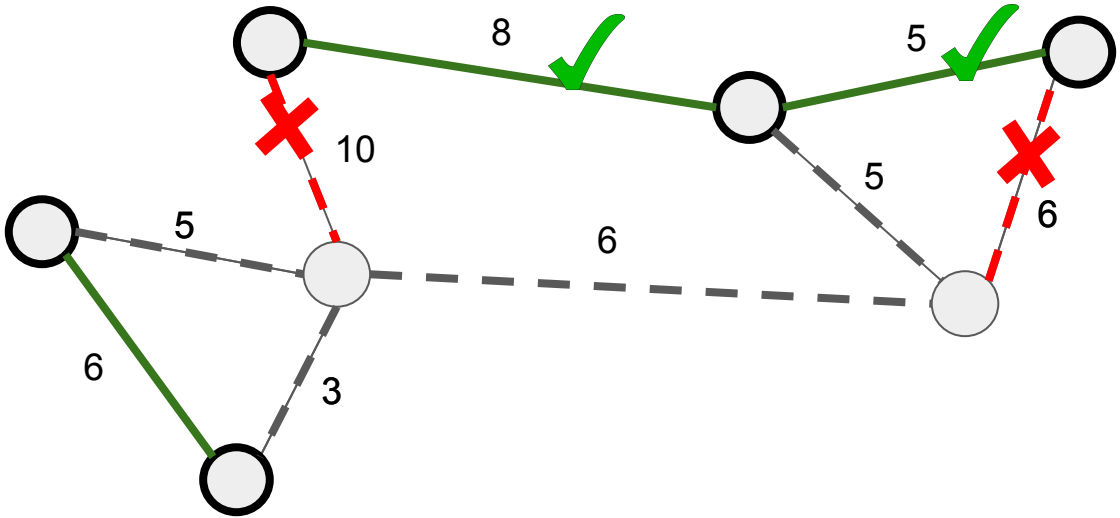
Batch-Incremental MST



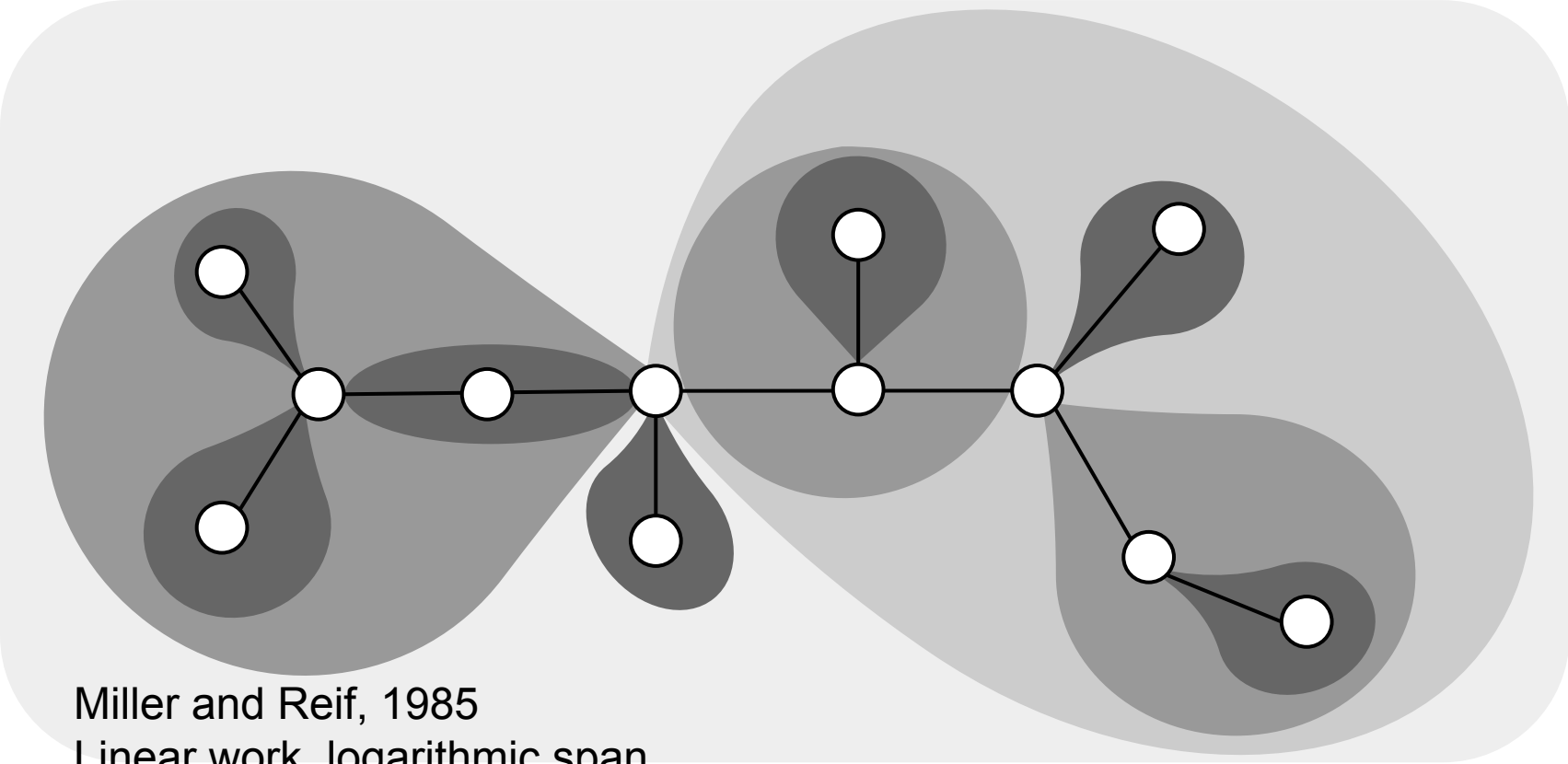
Batch-Incremental MST



Batch-Incremental MST

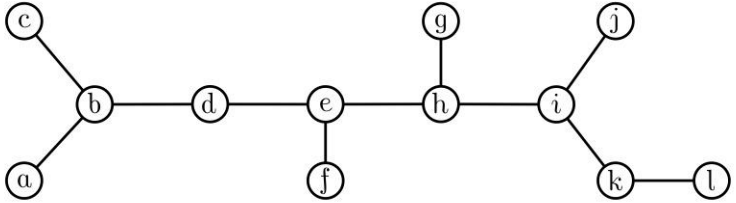


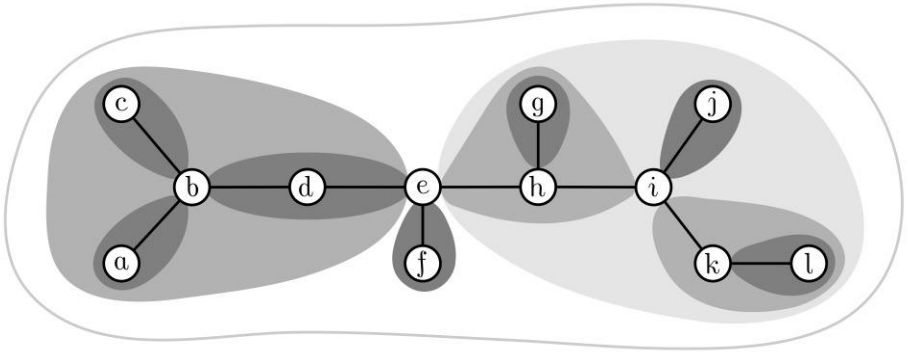
Tree Contraction and Rake-Compress Trees

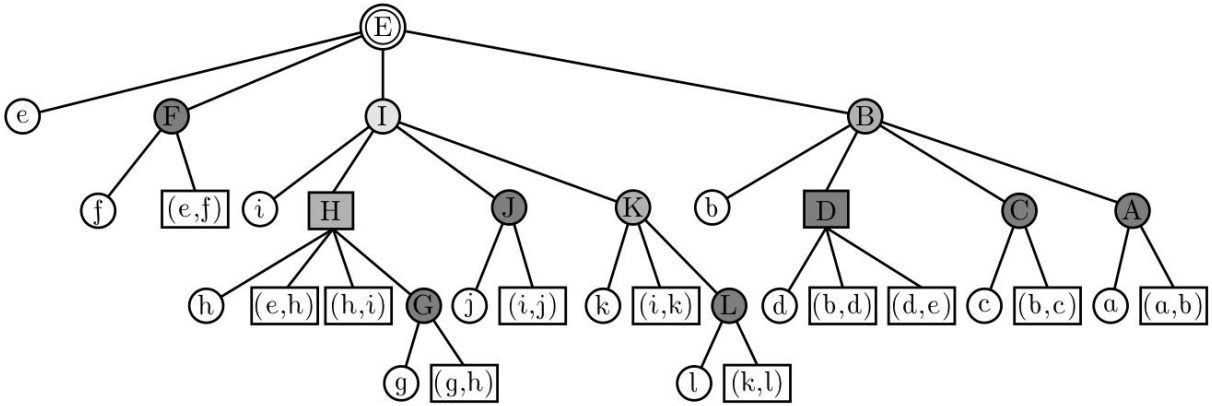


Miller and Reif, 1985
Linear work, logarithmic span

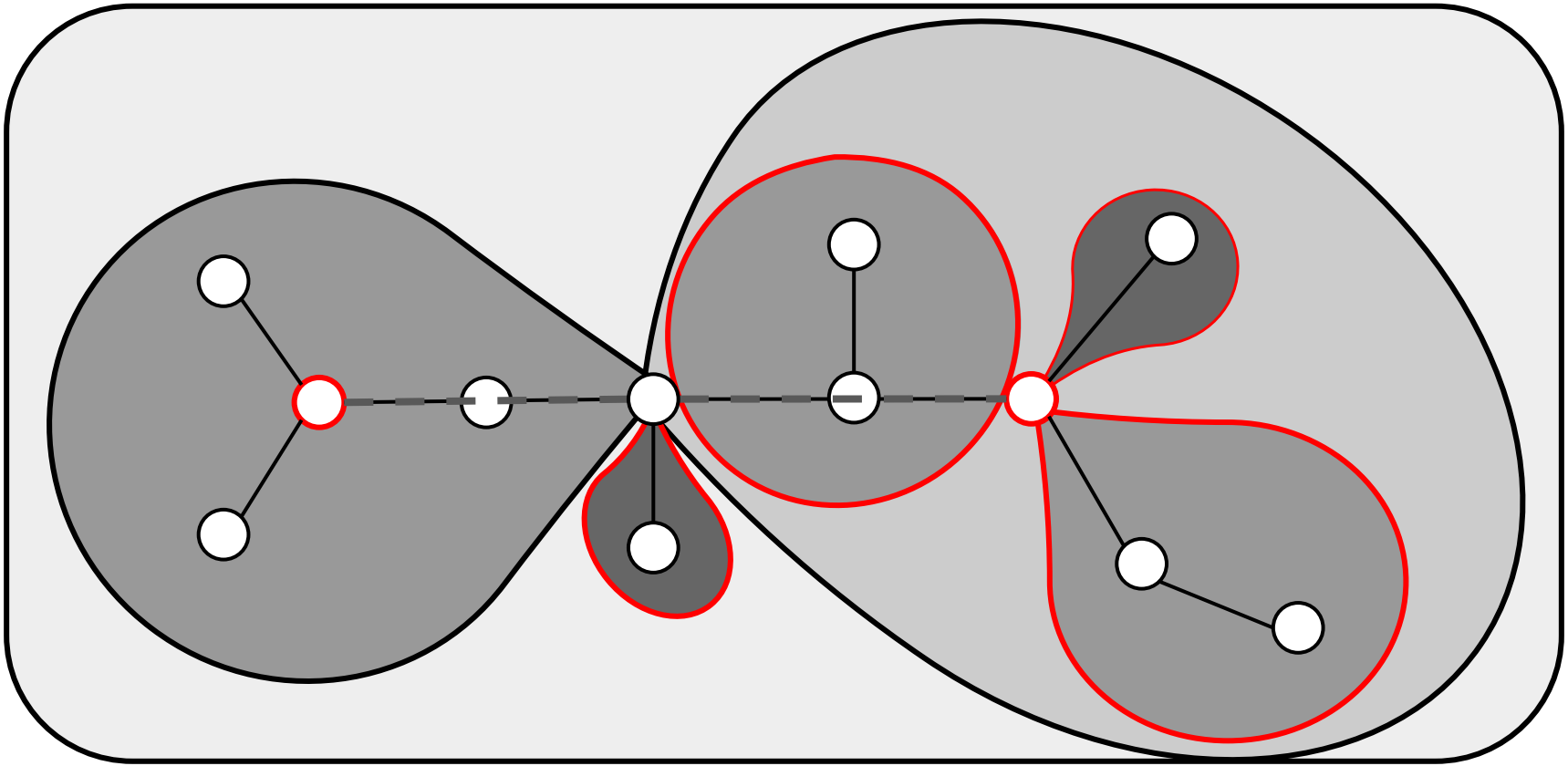
The RC Tree


c
b
d
e
g
h
i
j
a
f
k
l


c
b
d
e
g
h
i
j
a
f
k
l


E
e
F
I
B
f
(e,f)
i
H
J
K
b
D
C
A
h
(e,h)
(h,i)
G
j
(i,j)
k
(i,k)
L
d
(b,d)
(d,e)
c
(b,c)
a
(a,b)
g
(g,h)
l
(k,l)

Parallel Compressed Path Trees



Parallel Compressed Path Trees

High-level algorithm:

- Recursively traverse the Rake-compress tree top down
- Traverse into clusters that contain a marked vertex
- When a cluster with no marked vertices is encountered, it can either be deleted or immediately compressed down into a single edge

Using properties of Rake-compress trees, this takes $O(k \log(1 + n/k))$ work in $O(\log^2(n))$ span.

Summary: Parallel Batch-Incremental MST

To update an MST with a batch of edges:

- Compute the compressed path tree with respect to the new edges' endpoints
- Compute the MST of the compressed path tree and the new edges
- Insert all new edges that were selected by the MST and remove old edges that were not

Overall, this takes $O(k \log(1 + n/k))$ work in $O(\log^2(n))$ span.

This matches the sequential bound for small k, but is even better for large k.

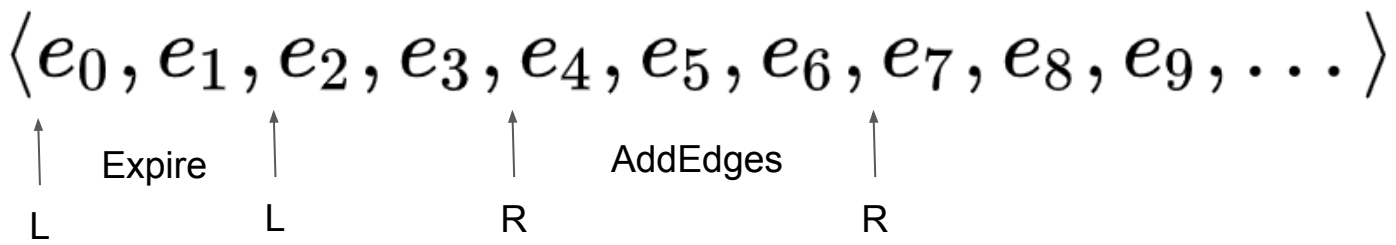
Applications of Incremental MST

Sliding-Window Dynamic Graph Algorithms

The sliding-window model assumes an infinite stream of edges $\langle e_0, e_1, e_2, \ldots \rangle$. We want to support the operations

- AddEdges(E): Add the given set of edges E to G
- Expire(k): Delete the oldest k edges from G

while supporting a variety of queries.



Sliding-Window Dynamic Graph Algorithms

We show that the following queries can be supported efficiently in parallel:

- Connectivity
- K-certificates
- Bipartiteness
- Cycle-freeness
- Approximate MSTs
- Sparsification

Mixed Batched Updates and Queries

Used in $O(m \log^2 n)$ Work, Polylog Span Min-cut Algorithm

Batched Mixed Queries and Updates

For an arbitrary tree consider sequence of mixed operations such as:

1. Min or sum of subtree (query)
2. Min or sum of path (query)
3. Add weight to path (update)
4. Add weight to subtree (update)

Goal is to evaluate result of all queries in the sequence, in batch, in parallel.

Generalizes and improve results of Geissmann and Gianinazzi, 2018.

RC simple operations

An implementation of a set of operations on an RC tree is **RC simple**, if:

1. Each RC node stores a value calculated from its children
2. All updates modify the value of a leaf and propagate values to the root
3. All queries traverse from leaf to root examining node and child values

Theorem: Given an RC tree on a tree of size n, any batch of k simple operations can be run in O(k log n) work and polylog span, using O(k) memory.

Basic idea: Timestamp all operations at the leaves, and then merge timestamped values and queries of children at all internal nodes.
O(k) work, O(log n) span per level of the tree. Only keep one level at a time.

Applications

1. Parallelize Lovett and Sandlund's simple $O(m \log^3 n)$ work min-cut algorithm
   a. Path-update, edge-query
2. Parallelizing 2-respecting cuts in $O(m \log n)$ work, leading to an $O(m \log^2 m)$ min-cut algorithm
3. Cool application to Karger's random contraction algorithm for approximate min-cut

Applications

1. Parallelize Lovett and Sandlund's simple $O(m \log^3 n)$ work min-cut algorithm
   a. Path-update, edge-query
2. Parallelizing 2-respecting cuts in $O(m \log n)$ work, leading to an $O(m \log^2 m)$ min-cut algorithm
3. Cool application to Karger's random contraction algorithm for approximate min-cut
   a. The algorithm contracts edges in random order (biased by weight), and keeps track of minimum weight of cut across a contracted edge.
   b. Simulating random order in parallel is easy using MST of edges with weight being their position in the order
   c. Keeping track of minimum weight cut is tricky, but can be implemented with two RC-simple operations: join-edge (update), add-weight-to-vertex (update), component-weight (query).

Conclusions and Opportunities

1. Batched parallel dynamic operations are **fun** and come in many flavors (dynamic, incremental, sliding window, mixed query and update)
2. In addition to allowing (work-efficient) parallelization, can reduce costs even in the sequential setting: i.e. O(k log (n/k)) vs. O(k log n).

If you have a problem you think would be amenable to this approach it would be great to talk to you.

* Thanks to Daniel Anderson for the animations, and many of the ideas