Parallel Dynamic Tree Contraction

via Self-Adjusting Computation

Umut Acar[1,2] Vitaly Aksenov[2,3] **Sam Westrick**[1]

[1]Carnegie Mellon University, USA

[2]Inria, France

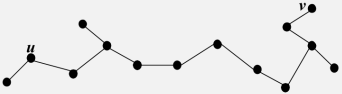
[3]ITMO University, Russia

July 2017

Introduction

Dynamic Algorithms

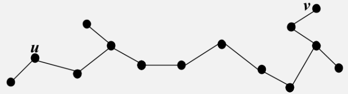
HeaviestEdgeBetween (



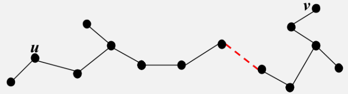
)

Introduction

Dynamic Algorithms

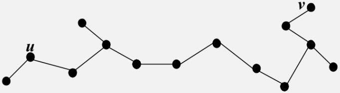
HeaviestEdgeBetween ()

modify input $\Downarrow$

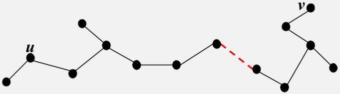
HeaviestEdgeBetween ()

Introduction

Dynamic Algorithms

HeaviestEdgeBetween $\left(\right.$  $\left. \right)$

modify input $\Downarrow$

HeaviestEdgeBetween $\left(\right.$  $\left. \right)$

Efficiency?

- Lots of work on **sequential algorithms** with **unit changes**.
  - e.g., single edge insertion/deletion
- More efficient to apply **many changes** simultaneously.

Contributions

Question

In a forest of size n, how efficiently can we recompute some desired property after applying a batch of m changes (insertions/deletions of edges/vertices)?

Contributions

Question

In a forest of size n, how efficiently can we recompute some desired property after applying a batch of m changes (insertions/deletions of edges/vertices)?

Results

	Work	Span
Construction	$O(n)$	$O(\log^2(n)))$
Update	$O\left(m \log \frac{n+m}{m}\right)$	$O(\log(n+m)\log(m))$

Contributions

Question

In a forest of size n, how efficiently can we recompute some desired property after applying a batch of m changes (insertions/deletions of edges/vertices)?

Results

	Work	Span
Construction	$O(n)$	$O(\log^2(n)))$
Update	$O\left(m \log \frac{n+m}{m}\right)$	$O(\log(n+m)\log(m))$

$$1 \longleftarrow m \longrightarrow n$$

$$\log n \longleftarrow m \log \frac{n+m}{m} \longrightarrow n$$

Contributions

Question

In a forest of size n, how efficiently can we recompute some desired property after applying a batch of m changes (insertions/deletions of edges/vertices)?

Results

	Work	Span
Construction	$O(n)$	$O(\log^2(n)))$
Update	$O\left(m \log \frac{n+m}{m}\right)$	$O(\log(n+m)\log(m)))$

$$1 \longleftarrow m \longrightarrow n$$

$$\log n \longleftarrow m \log \frac{n+m}{m} \longrightarrow n$$

(optimal) (optimal)

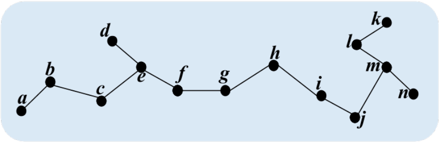
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

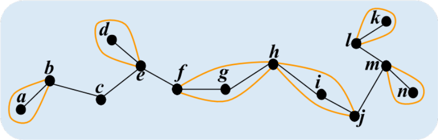
Round 0



Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

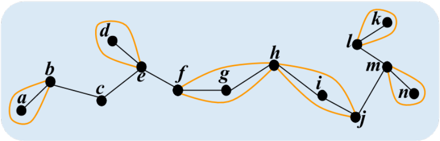
Round 0



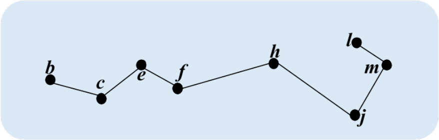
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

Round 0



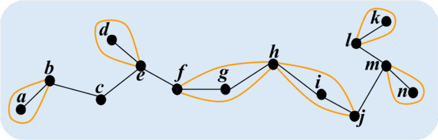
Round 1



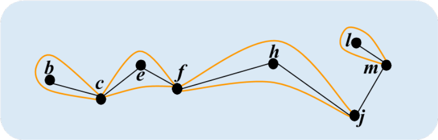
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

Round 0



Round 1



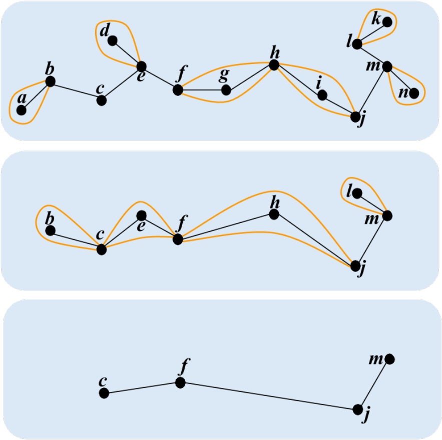
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

Round 0

Round 1

Round 2



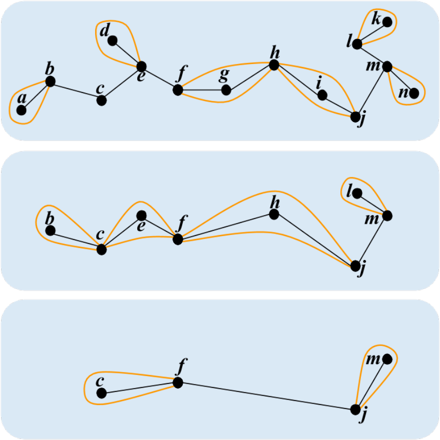
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

Round 0

Round 1

Round 2



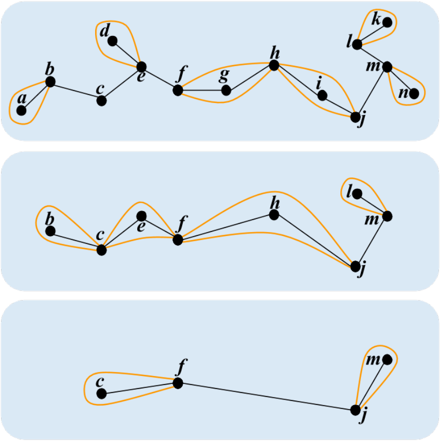
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- *rake* and *compress*
- $O(n)$ work
- $O(\log n)$ rounds

Round 0

Round 1

Round 2



⋮

Dynamizing Parallel Tree Contraction

$$\text{Algorithm} \xrightarrow{\textit{Dynamization}} \text{Dynamic Algorithm}$$

Dynamizing Parallel Tree Contraction

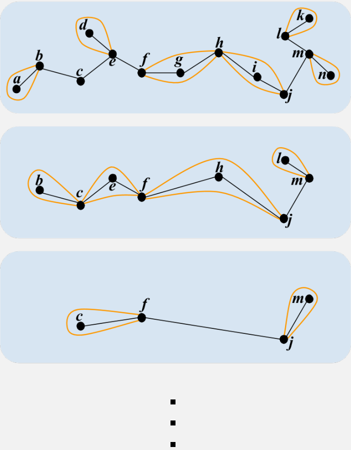
Parallel Algorithm $\xrightarrow{\textit{Dynamization}}$ **Parallel ?** Dynamic Algorithm

Dynamizing Parallel Tree Contraction

Parallel Algorithm $\xrightarrow{\textit{Dynamization}}$ **Parallel ✓** Dynamic Algorithm

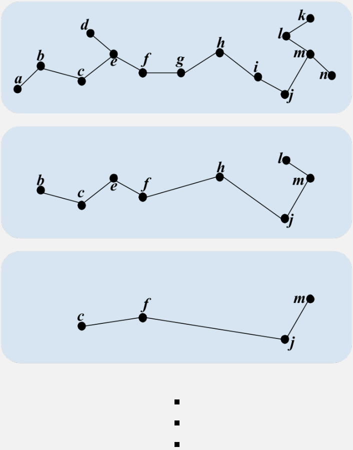
Dynamizing Parallel Tree Contraction

Construction



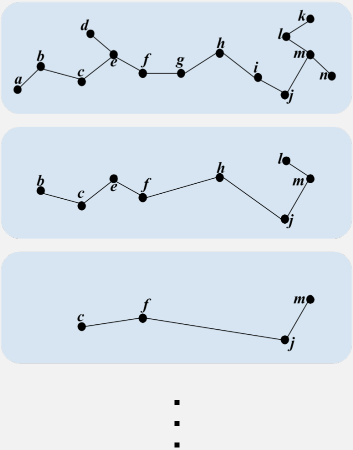
Dynamizing Parallel Tree Contraction

Construction

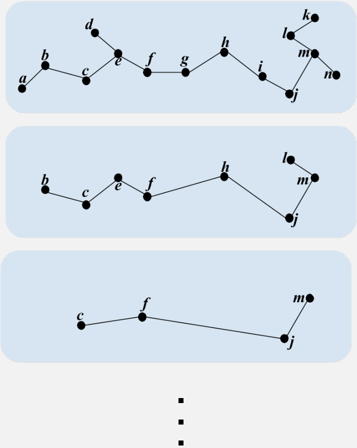


Dynamizing Parallel Tree Contraction

Construction

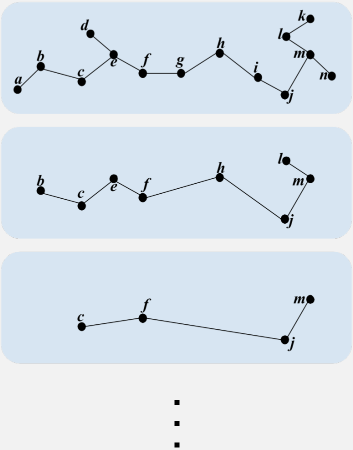


Update

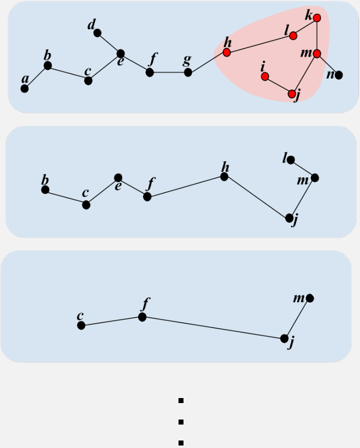


Dynamizing Parallel Tree Contraction

Construction

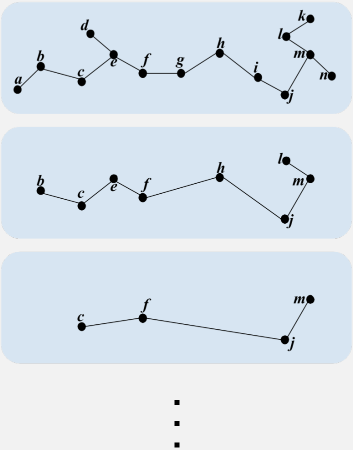


Update

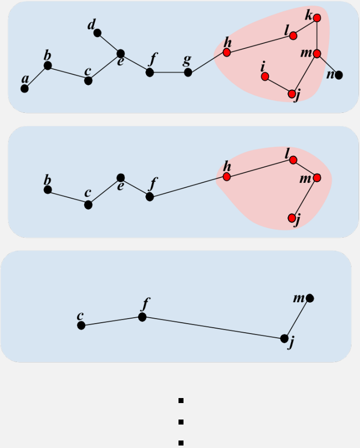


Dynamizing Parallel Tree Contraction

Construction

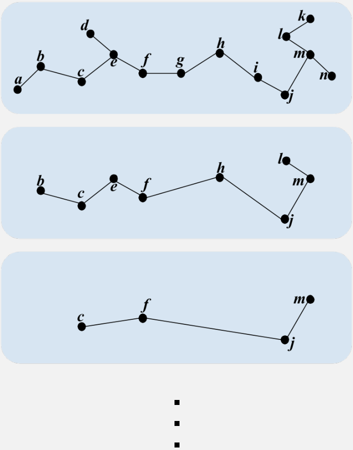


Update

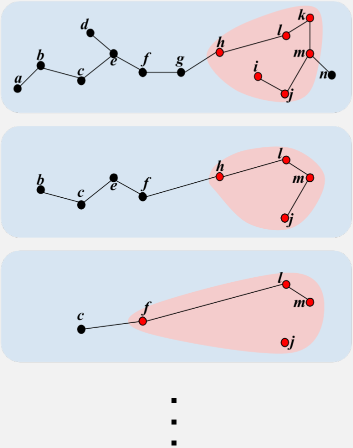


Dynamizing Parallel Tree Contraction

Construction



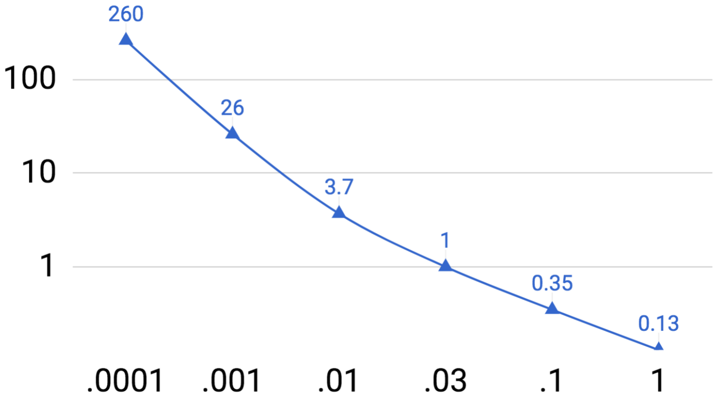
Update



Measuring Performance

Measuring Performance

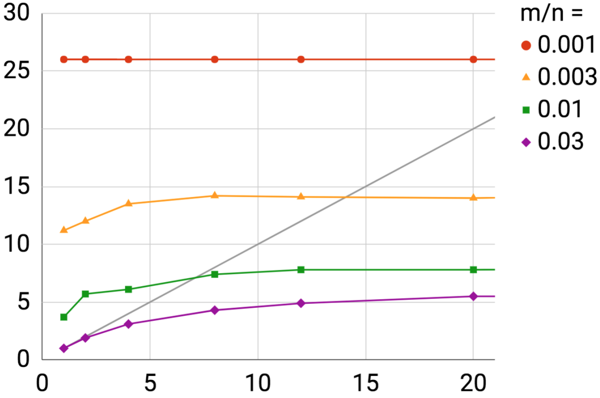
- Work Improvement: $\dfrac{T_{\text{static}}(\text{processors} = 1)}{T_{\text{update}}(\text{processors} = 1)}$
- Benefit of dynamism alone.



m/n (number of changes relative to forest size)

Measuring Performance

- Speedup: $\dfrac{T_{\text{static}}(\text{processors} = 1)}{T_{\text{update}}(\text{processors} = P)}$
- Combined benefit of dynamism and parallelism on P processors.



P (number of processors)

Conclusion

Summary

- We dynamized parallel tree contraction.
- The resulting algorithm is efficient both in theory and practice.

Closing Thoughts

- Some parallel algorithms are amenable to dynamization.
  - Take advantage of independent subproblems.
  - How many more examples are there?
- Are there general purpose techniques for *automatic* dynamization?

End

Thank You!

Questions?

Clustering Hierarchy