Parallel Dynamic Tree Contraction
via Self-Adjusting Computation

Umut Acar\textsuperscript{1,2} Vitaly Aksenov\textsuperscript{2,3} Sam Westrick\textsuperscript{1}

\textsuperscript{1}Carnegie Mellon University, USA
\textsuperscript{2}Inria, France
\textsuperscript{3}ITMO University, Russia

July 2017
Introduction

Dynamic Algorithms

HeaviestEdgeBetween($u$, $v$)

Efficiency? Lots of work on sequential algorithms with unit changes. More efficient to apply many changes simultaneously.

Umut Acar, Vitaly Aksenov, Sam Westrick (Carnegie Mellon University, USA, Inria, France, ITMO University, Russia)
Introduction

Dynamic Algorithms

HeaviestEdgeBetween

modify input \( \Downarrow \)

HeaviestEdgeBetween

Efficiency?
Lots of work on sequential algorithms with unit changes.

▶ e.g., single edge insertion/deletion
More efficient to apply many changes simultaneously.

Umut Acar, Vitaly Aksenov, Sam Westrick
Carnegie Mellon University, USA, Inria, France, ITMO University, Russia
Parallel Dynamic Tree Contraction
July 2017
Introducción

**Dynamic Algorithms**

HeaviestEdgeBetween

<table>
<thead>
<tr>
<th>modify input</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeaviestEdgeBetween</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction</td>
<td>$O(n)$</td>
<td>$O(\log^2(n))$</td>
</tr>
<tr>
<td>Update</td>
<td>$O\left(m \log \frac{n+m}{m}\right)$</td>
<td>$O(\log(n+m) \log(m))$</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction</td>
<td>$O(n)$</td>
<td>$O(\log^2(n))$</td>
</tr>
<tr>
<td>Update</td>
<td>$O\left( m \log \frac{n+m}{m} \right)$</td>
<td>$O(\log(n + m) \log(m))$</td>
</tr>
</tbody>
</table>

1 \[ \log n \] \[ m \] \[ m \log \frac{n+m}{m} \] \[ n \]
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

<table>
<thead>
<tr>
<th></th>
<th>Work</th>
<th>Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction</td>
<td>$O(n)$</td>
<td>$O(\log^2(n))$</td>
</tr>
<tr>
<td>Update</td>
<td>$O\left(m \log \frac{n+m}{m}\right)$</td>
<td>$O(\log(n + m) \log(m))$</td>
</tr>
</tbody>
</table>

$1 \quad \leftarrow \quad m \quad \longrightarrow \quad n$

$log n \quad \leftarrow \quad m \log \frac{n+m}{m} \quad \longrightarrow \quad n$

(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
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
Background: Parallel Tree Contraction

- Miller, Reif (1985)
- rake and compress
- $O(n)$ work
- $O(\log n)$ rounds
Dynamizing Parallel Tree Contraction

\[ \text{Algorithm} \longrightarrow \text{Dynamization} \longrightarrow \text{Dynamic Algorithm} \]
Dynamizing Parallel Tree Contraction

Parallel Algorithm \( \xrightarrow{\text{Dynamization}} \) Dynamic Algorithm

Parallel ?
Dynamizing Parallel Tree Contraction

Parallel Algorithm → Dynamization → Parallel Dynamic Algorithm
Dynamizing Parallel Tree Contraction

Construction

...
Dynamizing Parallel Tree Contraction

Construction

...
Dynamizing Parallel Tree Contraction

---

**Construction**

- Initial graph structure
- Step-by-step construction process

**Update**

- Initial graph structure
- Step-by-step update process
Dynamizing Parallel Tree Contraction

### Construction

- Initial tree
- Contracted tree
- Further contractions

### Update

- Initial tree with update
- Contracted tree with update
- Further contractions with update
Dynamizing Parallel Tree Contraction

Construction

Update
Dynamizing Parallel Tree Contraction

Construction

Update
Measuring Performance

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

\[ m/n \text{ (number of changes relative to forest size)} \]
Measuring Performance

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