An Improved Algorithm for Incremental DFS Tree in Undirected Graphs

Lijie Chen\textsuperscript{1}, Ran Duan\textsuperscript{2}, Ruosong Wang\textsuperscript{3}, Hanrui Zhang\textsuperscript{4}, \textbf{Tianyi Zhang}\textsuperscript{2}

\textsuperscript{1}MIT, \textsuperscript{2}Tsinghua University, \textsuperscript{3}CMU, \textsuperscript{4}Duke University
Definition: DFS tree

- Given an undirected graph $G = (V, E)$ with a designated root
- **DFS tree**: a maximal tree containing the root, where every non-tree edge connects an ancestor and a descendant

![A DFS tree](image1.png)

![Not a DFS tree](image2.png)
Definition: Incremental DFS tree

Data structure

- Maintain a DFS tree $T$ in graph $G$

Update operations

- **Input:** insert an edge/vertex to $G$
- **Output:** print all edges of $T$
**Example:** Incremental DFS tree

<table>
<thead>
<tr>
<th>Input: Updates to $G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>Output: Change in $T$</td>
</tr>
</tbody>
</table>
**Example**: Incremental DFS tree

**Input:**
Updates to $G$

**Output:**
Change in $T$

**Insert(3,4)**
Example: Incremental DFS tree

Input: Updates to $G$

Picture

Output: Change in $T$

Insert(3,4)

Delete(2,4)

Insert(3,4)
**Example: Incremental DFS tree**

<table>
<thead>
<tr>
<th>Input: Updates to $G$</th>
<th>Insert(3,4)</th>
<th>Insert(2,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Picture</strong></td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>Output:</strong> Change in $T$</td>
<td>Delete(2,4) Insert(3,4)</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image3)
Example: Incremental DFS tree

Input: Updates to $G$

Output: Change in $T$

<table>
<thead>
<tr>
<th>Picture</th>
<th>Insert(3,4)</th>
<th>Insert(2,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
<td><img src="image3.png" alt="Graph" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insert(3,4)</th>
<th>Delete(2,4)</th>
<th>Insert(3,4)</th>
<th>Insert(2,5)</th>
<th>Delete(1,5)</th>
<th>Insert(2,5)</th>
</tr>
</thead>
</table>
**Example: Incremental DFS tree**

**Input:** Updates to $G$

**Output:** Change in $T$

<table>
<thead>
<tr>
<th>Picture</th>
<th>Insert(3,4)</th>
<th>Insert(2,5)</th>
<th>Insert(4,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="" alt="Diagram" /></td>
<td><img src="" alt="Diagram" /></td>
<td><img src="" alt="Diagram" /></td>
<td><img src="" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **Insert(3,4)**: The edge between nodes 3 and 4 is added.
- **Delete(2,4)**: The edge between nodes 2 and 4 is removed.
- **Delete(1,5)**: The edge between nodes 1 and 5 is removed.
- **Insert(3,4)**: The edge between nodes 3 and 4 is added.
- **Insert(2,5)**: The edge between nodes 2 and 5 is added.
- **Insert(4,5)**: The edge between nodes 4 and 5 is added.
### Example: Incremental DFS tree

**Input:** Updates to $G$

<table>
<thead>
<tr>
<th>Input: Updates to $G</th>
<th>Insert(3,4)</th>
<th>Insert(2,5)</th>
<th>Insert(4,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture</td>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Output:** Change in $T$

<table>
<thead>
<tr>
<th>Output: Change in $T</th>
<th>Delete(2,4)</th>
<th>Insert(3,4)</th>
<th>Delete(1,5)</th>
<th>Insert(2,5)</th>
<th>Insert(4,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture</td>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
<td><img src="image7.png" alt="Diagram" /></td>
<td><img src="image8.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Progress on incremental DFS

- $n = \# \text{ of vertices}, \ m = \# \text{ of edges}$

<table>
<thead>
<tr>
<th>Reference</th>
<th>Naïve</th>
<th>[BK’14]</th>
<th>[BCC+’16]</th>
<th>[NS’17]</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update time</td>
<td>$O(m + n)$</td>
<td>$O(n)$</td>
<td>$O(n \log^3 n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(m + n)$</td>
<td>$O(m + n)$</td>
<td>$O(m \log n)$</td>
<td>$O(m \log n)$ in bits</td>
<td>$O(m \log n)$</td>
</tr>
<tr>
<td>Worst-case?</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Reduction to batch insertion

Main Theorem:

- Preprocess graph $G$ in $O(\min\{m \log n, n^2\})$ time
- **Input:** a set $U$ of $k$ edge insertions
- **Output:** a DFS tree of $G+U$ in $O(n + k)$
Batch insertions [BCC+’16]

Revert & reroot
Batch insertions [BCC+’16]

Revert & reroot

new insertion
Batch insertions [BCC+’16]

Revert & reroot

revert this tree path

new insertion
Batch insertions [BCC+’16]

Revert & reroot

revert this tree path

new insertion

reroot
Batch insertions [BCC+’16]

Revert & reroot

How to relocate this subtree?
Batch insertions [BCC+’16]

Revert & reroot

How to relocate this subtree?
Find the highest ancestor…
Batch insertions [BCC+’16]

Revert & reroot

How to relocate this subtree?
Find the highest ancestor and reroot again.
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+'16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

Recursively revert & reroot

Diagram of batch insertions with nodes and edges.
Batch insertions [BCC+’16]

Recursively *revert* & *reroot*
Batch insertions [BCC+’16]

Recursively revert & reroot
Batch insertions [BCC+’16]

**Bottleneck:** finding highest ancestors on reverted paths
Batch insertions [BCC+’16]

Bottleneck: finding highest ancestors on reverted paths
Batch insertions [BCC+’16]

**Bottleneck:** finding *highest ancestors* on reverted paths
Batch insertions [BCC+’16]

**Bottleneck:** finding *highest ancestors* on reverted paths
Batch insertions [BCC+’16]

**Bottleneck:** finding *highest ancestors* on reverted paths
Batch insertions [BCC+’16]

**Bottleneck:** finding highest ancestors on reverted paths
Finding highest ancestors

**Tool I:** 2D-range query
Finding highest ancestors

Tool I: 2D-range query
Finding highest ancestors

**Tool I: 2D-range query**
Finding highest ancestors

**Tool I:** 2D-range query
Finding highest ancestors

**Tool I:** 2D-range query
Finding highest ancestors

**Tool I:** 2D-range query
Finding highest ancestors

**Tool I: 2D-range query**
Finding highest ancestors

**Tool I:** 2D-range query
Finding highest ancestors

**Tool I:** 2D-range query
Finding highest ancestors

**Tool I:** 2D-range query

- Euler-tour order
- DFS order
Finding highest ancestors

**Tool I:** 2D-range query

2D-range minimum takes $O(\log n)$ time
Finding highest ancestors

**Tool I:** 2D-range query

2D-range minimum takes $O(\log n)$ time

Total time ’d be $O(k + n \log n)$
Finding highest ancestors

**Tool II**: Fractional cascading & tree partitioning
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [CG’86]

Given \( k \) sorted integer arrays of total size \( m \)

- **Input:** integer \( x \)
- **Output:** successors of \( x \) in each array, in time \( O(k + \log m) \)
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [CG’86]

Given *k* sorted integer arrays of total size *m*

- **Input:** integer *x*
- **Output:** successors of *x* in each array, in time $O(k + \log m)$
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [CG’86]

Given $k$ sorted integer arrays of total size $m$

- **Input:** integer $x$
- **Output:** successors of $x$ in each array, in time $O(k + \log m)$
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [CG’86]

Given \( k \) sorted integer arrays of total size \( m \)

- **Input:** integer \( x \)
- **Output:** successors of \( x \) in each array, in time \( O(k + \log m) \)
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [DZ’17]

Given a tree $T$ of size $n$,

- **Input:** integer $k$
- **Output:** remove $O(n/k)$ special vertices to partition $T$ into subtrees of size at most $k$
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [DZ’17]

Given a tree $T$ of size $n$,

- **Input:** integer $k$
- **Output:** remove $O(n/k)$ special vertices to partition $T$ into subtrees of size at most $k$
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [DZ’17]

Given a tree $T$ of size $n$,

- **Input:** integer $k$
- **Output:** remove $O(n/k)$ special vertices to partition $T$ into subtrees of size at most $k$
Finding highest ancestors

**Tool II:** Fractional cascading & tree partitioning

**Lemma:** [DZ’17]

Given a tree $T$ of size $n$,

- **Input:** integer $k$
- **Output:** remove $O(n/k)$ special vertices to partition $T$ into subtrees of size at most $k$
Finding highest ancestors

New data structure for finding highest ancestors
Finding highest ancestors

New data structure for finding highest ancestors

Apply tree partition with $k = \log n$
So every subtree has size $O(\log n)$
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with $k = \log n$

So every **subtree** has size $O(\log n)$
Finding highest ancestors

New data structure for finding highest ancestors

Apply tree partition with $k = \log n$
So every subtree has size $O(\log n)$

Build fractional cascading on a bunch of subtrees
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with \( k = \log n \)
So every **subtree** has size \( O(\log n) \)

Build **fractional cascading** on a bunch of **subtrees**
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with \( k = \log n \)

So every **subtree** has size \( O(\log n) \)

Precompute for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply tree partition with \( k = \log n \)
So every subtree has size \( O(\log n) \)

Precompute for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with \( k = \log n \)
So every **subtree** has size \( O(\log n) \)

Precompute for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with \( k = \log n \)

So every **subtree** has size \( O(\log n) \)

**Precompute** for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with $k = \log n$
So every **subtree** has size $O(\log n)$

**Precompute** for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply tree partition with $k = \log n$
So every subtree has size $O(\log n)$

Precompute for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with \( k = \log n \)

So every **subtree** has size \( O(\log n) \)

**Precompute** for every choice of reverted tree path below the nearest special ancestor
Finding highest ancestors

New data structure for finding highest ancestors

Apply **tree partition** with $k = \log n$
So every **subtree** has size $O(\log n)$

**Precompute** for every choice of reverted tree path below the nearest special ancestor

$O(n \log n)$ **entries in total**
Highest ancestors in $O(n)$ total time

Consider three cases below
Highest ancestors in $O(n)$ total time

Consider three cases below

The reverted tree path contains no special vertices
Highest ancestors in $O(n)$ total time

Consider three cases below

The reverted tree path contains no special vertices

Use precomputed entries $O(1)$ time
Highest ancestors in $O(n)$ total time

Consider three cases below

Use precomputed entries $O(1)$ time

Total time = $O(n)$

The reverted tree path contains no special vertices
Highest ancestors in $O(n)$ total time

Consider three cases below

The reverted tree path contains a special vertex, and subtree-size > $log n$
Highest ancestors in $O(n)$ total time

Consider three cases below

The reverted tree path contains a special vertex, and subtree-size $> \log n$

Apply 2D-range minimum $O(\log n)$ time
Highest ancestors in $O(n)$ total time

Consider three cases below

The reverted tree path contains a special vertex, and subtree-size > $\log n$

Apply 2D-range minimum $O(\log n)$ time

One can prove this happens at most $O(n / \log n)$ times, so total time becomes $O(n)$
Highest ancestors in $O(n)$ total time

Consider three cases below

A bunch of subtrees containing no special vertices
Highest ancestors in $O(n)$ total time

Consider three cases below

A bunch of subtrees containing no special vertices

Apply fractional-cascade $O(\text{subtree-sizes} + \log n)$ time
Highest ancestors in $O(n)$ total time

Consider three cases below

A bunch of subtrees containing no special vertices

Apply fractional-cascade $O(\text{subtree-sizes} + \log n)$ time

Sum of subtree-sizes $= O(n)$
Highest ancestors in $O(n)$ total time

Consider three cases below

Apply fractional-cascade $O(\text{subtree-sizes} + \log n)$ time

Sum of subtree-sizes = $O(n)$

One can prove this happens at most $O(n / \log n)$ times, so total time becomes $O(n)$

A bunch of subtrees containing no special vertices
Highest ancestors in $O(n)$ total time

Consider three cases below:

- A bunch of subtrees containing no special vertices

Apply fractional-cascade $O(\text{subtree-sizes} + \log n)$ time

- Sum of subtree-sizes = $O(n)$

- One can prove this happens at most $O(n / \log n)$ times, so total time becomes $O(n)$

Summary: total time is $O(n)$
Thanks