The Dynamic Trees Problem

- Dynamic trees:
  - Goal: maintain an \( n \)-vertex forest that changes over time.
    - \( \text{link}(v, w) \): creates an edge between vertices \( v \) and \( w \).
    - \( \text{cut}(v, w) \): deletes edge \( (v, w) \).
  - Application-specific data associated with edges and/or vertices.

- Concrete examples:
  - Find minimum-weight edge in the path between any two vertices.
  - Add a value to all edges in the path between two vertices.
  - Find total weight of all vertices in a subtree.

- \( O(\log n) \) time per operation.
  - Map arbitrary tree onto balanced tree.

Dynamic Trees

Data Structures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary subtree queries?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Arbitrary path queries?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Simple to implement?</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Generic interface?</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>( O(\log n) ) worst case?</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Principle</td>
<td>path decompress.</td>
<td>path decompress.</td>
<td>tree contraction</td>
<td>tree contraction</td>
<td>tree contraction</td>
<td>linearization (Euler tour)</td>
</tr>
</tbody>
</table>

Contractions: Rake and Compress

- Proposed by Miller and Reif [1985] (parallel setting).
- \textbf{Rake}:
  - Eliminates a degree-one vertex.
  - Collapses edge onto successor.
    - Assumes circular order of edges.
- \textbf{Compress}:
  - Eliminates a degree-two vertex.
  - Combines two edges into one.

Original edges and resulting edge are \textit{clusters}.

Contractions: Rake and Compress

- \textbf{Rake}:
  - Series of rakes and compresses;
  - Reduces a tree to a single cluster (edge).
- \textbf{Compress}:
  - Direct access only to root cluster.
  - User defines what information to store in parent.
  - Any order of rakes and compresses is "right":
    - root will have the correct information.
    - Balanced: updates in \( O(\log n) \) time.
    - Alstrup et al. [1997] use topology trees: high overhead.
- We show a direct implementation.
• Consider some unrooted tree:

• Pick a degree-one vertex as root, direct all edges towards it.
  • We call this a unit tree (rooted tree with degree-one root).

• Pick a root path:
  • starts at some leaf;
  • ends at the root.

• Represent the root path as a binary tree:
  • Leaves: base clusters (original edges).
  • Internal nodes: compress clusters.

• What if the degree of a vertex is not two?
  • Recursively represent each subtree rooted at the vertex.
    • At most two because of circular order.

• What if the degree of a vertex is not two?
  • Recursively represent each subtree rooted at the vertex.
  • Before vertex is compressed, rake subtree onto adjacent cluster.
Dynamic Trees

• Representation:
  - Up to four children per node (up to two foster children).
  - Meaning: up to two rakes followed by a compress.

• Example: \(N_e = \text{compress}(\text{rake}(X, ce), \text{rake}(Z, ef)) = cf\)

• How does the recursive representation work?
  - Must represent subtrees rooted at the root path.

• How does the recursive representation work?
  - Must represent subtrees rooted at the root path.
  - Each subtree is a sequence of unit trees.

• How does the recursive representation work?
  - Must represent subtrees rooted at the root path.
  - Each subtree is a sequence of unit trees.
  - Represent each unit tree recursively.

• How does the recursive representation work?
  - Must represent subtrees rooted at the root path.
  - Each subtree is a sequence of unit trees.
  - Represent each unit tree recursively.

• Interpretations:
  - User interface: tree contraction.
  - A sequence of rakes and compresses;
  - A single tree;
  - Similar to topology trees and RC-trees.

• Implementation: path decomposition.
  - Maximal edge-disjoint paths;
  - Hierarchy of binary trees (rake trees/compress trees);
  - Similar to ST-trees.
Self-Adjusting Top Trees

- Topmost compress tree represents the root path.
  - Top tree interface allows the user to access the root path only.
  - \( \text{expose} \) makes a node \( v \) part of the root path (and/or changes root).
    - Main tools: splay and splice.

- Splicing: changes the partition of the original tree into paths.
  - \( \text{expose}(v) \) in 3 passes:
    1. Splay within each binary tree between \( v \) and the root;
    2. perform a series of \( \text{splices} \);
    3. splay within the final tree.
  - Main result: \( O(\log n) \) amortized time.

Links

- \( \text{link}(v, w) \): first expose \( v \) and \( w \), then rearrange appropriately.

Hidden Details

- Exposing the vertex is slightly different from changing the root.
- Top tree nodes represent edges; must also associate with vertices.
- Degree of vertices exposed matters (special cases).
- Left-right relation must be relaxed in compress trees.
- Must call user-defined functions in the appropriate order.
### Practical Considerations

- **Compress node:**
  - Actually represents up to 3 clusters.
  - Could be implemented as one cluster = one node.
    - Splaying and splicing get slightly more complicated.
- **Special cases (application-dependent):**
  - No circular order:
    - Compress nodes have at most 3 (not 4) children.
    - Simpler splices.
  - Trivial rakes: essentially ST-Trees.
    - No rake trees.
    - No pointers to "middle children" (dashed edges).

### Further Work

- **Worst-case variant?**
- **Careful experimental study:**
  - Top trees tend to be slower than ET-trees and ST-trees, but:
    - More generic:
      - bounded/unbounded degrees;
      - subtree/path operations;
      - circular order around vertices.
    - Much easier to adapt to different applications;
    - Easier to reason about.
  - How does it compare to RC-trees?