Self-Adjusting Top Trees

Robert Tarjan (Princeton University/HP)

Renato Werneck (Princeton University)

The Dynamic Trees Problem

- Dynamic trees:
  - Goal: maintain an *n*-vertex forest that changes over time.
    - link(*v*,*w*): creates an edge between vertices *v* and *w*.
    - 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.

Data Structures

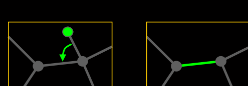
	ST-trees [ST83]	ST-trees [ST85]	Topology [Fre85]	RC-trees [ABHW03]	Top Trees [AHdLT97]	ET-trees [HK95]
Arbitrary subtree queries?	bounded degree	bounded degree	bounded degree	bounded degree	YES	YES
Arbitrary path queries?	YES	YES	bounded degree	bounded degree	YES	NO
Simple to implement?	NO	fairly	fairly	fairly	interface only	YES
Generic interface?	NO	NO	NO	YES	YES	NO
O(log n) worst case?	YES	amortized	YES	randomized	interface only	YES
Principle	path decomp.	path decomp.	tree contraction	tree contraction	tree contraction	linearization (Euler tour)

Data Structures

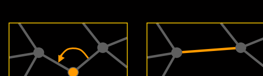
	ST-trees [ST83]	ST-trees [ST85]	Topology [Fre85]	RC-trees [ABHW03]	Top Trees [AHdLT97]	ET-trees [HK95]
Arbitrary subtree queries?	bounded degree	bounded degree	bounded degree	bounded degree	YES	YES
Arbitrary path queries?	YES	YES	bounded degree	bounded degree	YES	NO
Simple to implement?	NO	fairly	fairly	fairly	fairly	YES
Generic Interface?	NO	NO	NO	YES	YES	NO
O(log n) worst case?	YES	amortized	YES	randomized	amortized	YES
Principle	path decomp.	path decomp.	tree contraction	tree contraction	tree contr./ path decomp.	linearization (Euler tour)

Contractions: Rake and Compress

- Proposed by Miller and Reif [1985] (parallel setting).
- Rake:
  - Eliminates a degree-one vertex.
  - Collapses edge onto successor.
    - Assumes circular order of edges.



- Compress:
  - Eliminates a degree-two vertex.
  - Combines two edges into one.



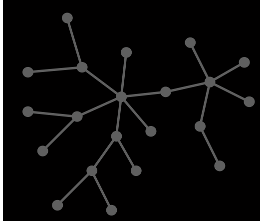
- Original edges and resulting edge are clusters.

Contractions: Rake and Compress

- Contraction:
  - Series of rakes and compresses;
  - Reduces a tree to a single cluster (edge).
- *Top tree* embodies a contraction:
  - 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.

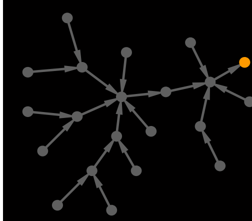
Representation

- Consider some unrooted tree:



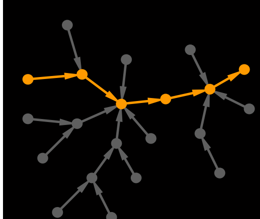
Representation

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



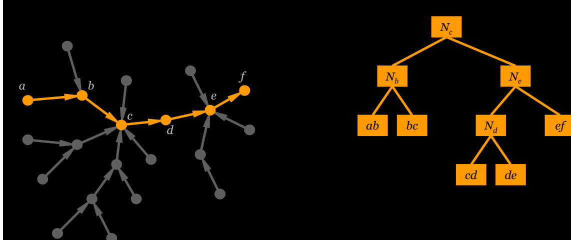
Representation

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



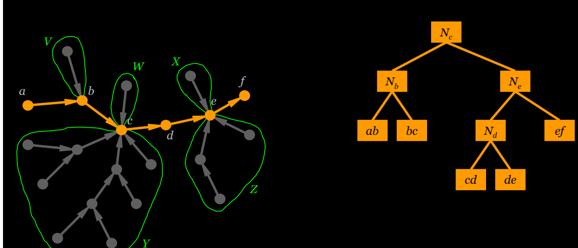
Representation

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



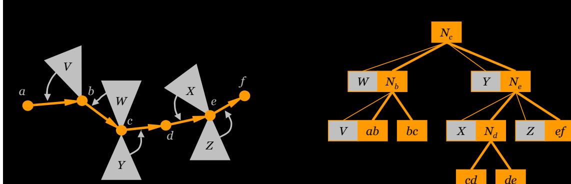
Representation

- 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.



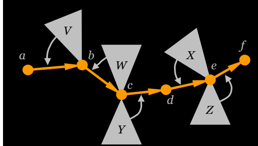
Representation

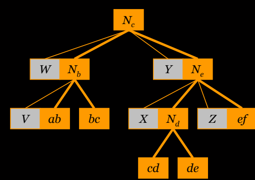
- What if the degree of a vertex is not two?
  - Recursively represent each subtree rooted at the vertex.
  - Before vertex is compressed, rake subtree onto adjacent cluster.



Representation

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

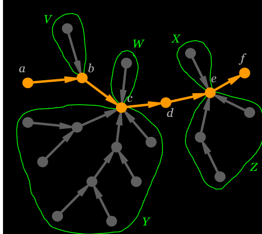




  - Example: N_e = compress(rake(X, ce), rake(Z, ef)) = cf

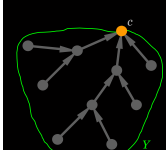
Representation

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



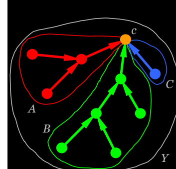
Representation

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



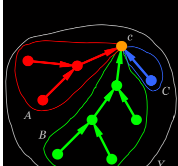
Representation

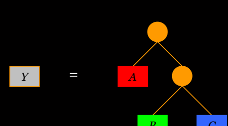
- How does the recursive representation work?
  - Must represent subtrees rooted at the root path.
  - Each subtree is a sequence of unit trees.
  - Represent each unit tree recursively.



Representation

- 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.
  - Build a binary tree of rakes.





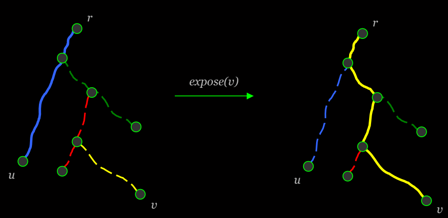
(Each circle is rake cluster.)

Representation

- Interpretations:
  - User interface: tree contraction.
    - 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.

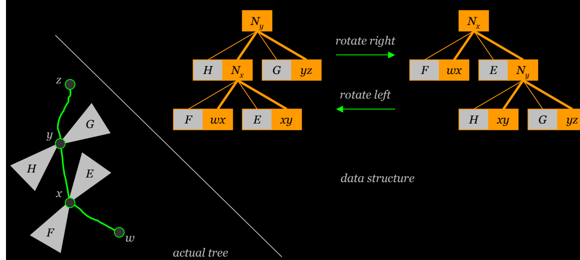
Self-Adjusting Top Trees

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



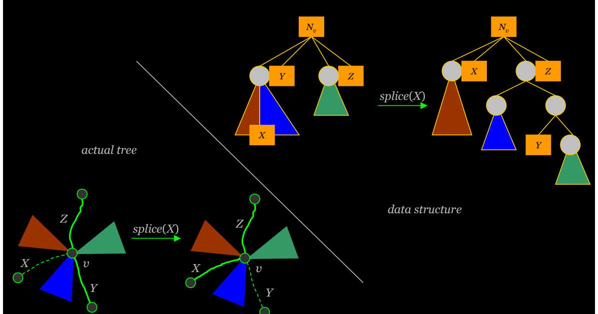
Self-Adjusting Top Trees

- Splaying: series of rotations within a rake/compress subtree:
  - keeps subtree "balanced" (in the amortized sense);
  - brings vertex to the root of the subtree.



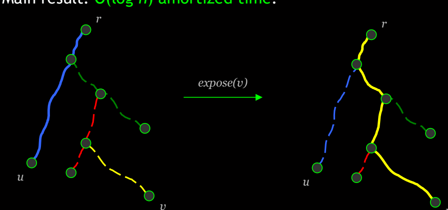
Self-Adjusting Top Trees

- Splice: changes the partition of the original tree into paths.



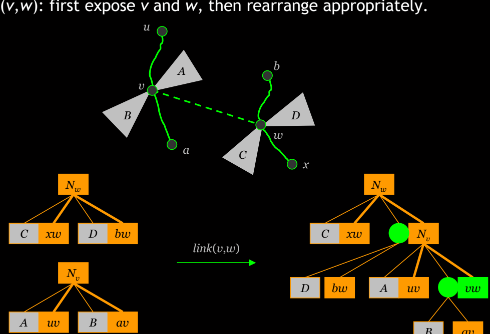
Self-Adjusting Top Trees

- expose(*v*) in 3 passes:
  1. Splay within each binary tree between *v* and the root;
  2. perform a series of *splices*;
  3. splay within the final tree.
- Main result: O(log *n*) amortized time.



Links

- 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?