Decremental Connectivity in Trees

Philip Bille
Outline

• Decremental Connectivity in Trees Problem
• Decremental Connectivity in Paths
  • First Tradeoffs
  • Two-Level Solution
• Decremental Connectivity in Trees
  • Two-Level Solution
Decremental Connectivity Problem
Decremental Connectivity Problem

• The **decremental connectivity problem**: Starting with a tree $T$ with $n$ nodes support the following operations:
  
  • `connected(v,u)`: return true if $v$ and $u$ are connected.
  
  • `delete(e)`: delete the edge $e$. 
Applications

• Special case of dynamic graph algorithm (inserting and deleting edges/nodes) while supporting some query/queries.

• Nice illustration of techniques for trees and word-level parallelism.

• Nice illustration of algorithmic theory useful in practice.
Overview

• First consider the simple case of paths. Later generalize to trees.

• Goal:
  • $O(1)$ for connected.
  • $O(n)$ total for executing all $n-1$ delete operations.
Overview

• Solution in 3 steps:
  • First tradeoffs: queries vs. updates
  • Balanced relabeling
  • Clustering with word-level parallelism
First Tradeoffs
First Tradeoffs

- What tradeoffs can we get for connected queries vs. deletions?
Fast Deletions

• How fast deletions can we get if we ignore the time for connected?
Fast Deletions

• Solution:
  • delete(e): remove e from path.
  • connected(v,u): traverse component from v to see if u is in component containing v.

• O(1) for deletion => O(n) for n-1 deletions
• O(n) for connected
Fast Connected Queries

• How fast connected queries can we get if we ignore the time for deletions?
Fast Connected Queries

- Solution: Maintain a component ID for each node.
- \( v \) and \( u \) are connected iff \( ID(v) = ID(u) \).
  - \text{connected}(v,u): return true iff \( ID(v) = ID(u) \)
  - \text{delete}(e): relabel node IDs of left endpoint of \( e \) (or right endpoint of \( e \)).
- \( O(1) \) for \text{connected}
- \( O(n) \) for single delete \( \Rightarrow O(n^2) \) for \( n-1 \) deletions
Better Tradeoffs?

• We have two extreme tradeoffs:
  • connected O(n) and n-1 deletes O(n).
  • connected O(1) and n-1 deletes O(n^2).
• How can we get better tradeoffs?
• Consider the bad case for relabeling algorithm.
A Bad Delete Sequence

- Main problem: n-1 delete operations take
  - $O((n-1) + (n-2) + \cdots + 1) = O(n^2)$
- How can we do better?
- Idea: After a delete update ID for the smaller component.
Balanced Relabeling

- New delete(e):
  - Traverse components for endpoints of e \textit{in parallel}.
  - Stop when we find the \textit{smallest} component.
  - Update ID for smallest component.

- ID(v) is only updated when v is in the smaller component.
- \( \Rightarrow \) After first update to ID(v) the size of v’s component is \( \leq n/2 \), next \( \leq n/4 \), next \( \leq n/8 \),...
- \( \Rightarrow \) At most log n updates for ID(v).
- Total time for \( n-1 \) deletions = \( O(\text{Total number of ID updates}) = O(n \log n) \).
Summary

• Theorem: We can solve decremental connectivity in paths in
  • $O(1)$ time for connected.
  • $O(n \log n)$ time for $n-1$ deletes.
• How can we get rid of the log-factor?
Shaving a Log-Factor
Overview

• Goal:
  • O(1) time connected
  • O(n) time for n-1 deletes.

• Solution by two-level data structure:
  • Divide path into n/w subpath of w nodes.
  • Level 1: Balanced relabeling data structure over path of n/w nodes.
  • Level 2: New data structure for paths of length ≤ w supporting delete and connected in O(1) time.
Data Structure for Short Paths
Paths of length $\leq w$.

Data structure consists of bitmasks:

- $C$: $C[i] = 1$ iff edge $i$ exists.
- For each node $v$:
  - $R(v)$: $R(v)[i] = 1$ iff $i$ is to the right of $v$.
  - $L(v)$: $L(v)[i] = 1$ iff $i$ is to the left of $v$.
- $O(w)$ bitmasks of length $\leq w \Rightarrow O(w)$ space.
Delete

- How can we implement delete in O(1) time?
- \text{delete(edge } i\text{): set } C[i] = 0.
Connected

- How can we implement connected in $O(1)$ time?
- $\text{connect}(v,u) = \text{Return true iff } R(v) \& L(u) \& \neg C = 0$
Summary

• Theorem: We can solve decremental connectivity in paths of length $\leq w$ in
  • $O(1)$ time for connected.
  • $O(1)$ time for delete.
Two-Level Data Structure
A Two-Level Solution

- Divide path into $n/w$ subpath of $w$ nodes.
- Level 1: The $O(n/w)$ boundary nodes of the subpaths. Edge between two boundary nodes iff connected by subpath. Maintain using balanced relabeling data structure.
- Level 2: Short path data structure for each subpath.
A Two-Level Solution

- delete(e): delete edge in level 2. If first deletion in subpath also delete in level 1 using balanced relabeling strategy.

- connected(v,u):
  - Case 1: v and u in same subpath. Use level 2 data structure.
  - Case 2: v and u in different subpaths. Return true iff
    - connected(v, right-boundary(v)) &
    - connected(u, left-boundary(u)) &
    - connected(right-boundary(v), left-boundary(u))
A Two-Level Solution

- O(1) time for connected (at most 1 connected query in level 1 + 2 connected queries in level 2)

- n-1 delete operations:
  - Level 2: O(n)
  - Level 1: O(n/w \cdot \log (n/w)) = O(n)

- => O(n) in total.
Summary

- Theorem: We can solve decremental connectivity in paths in
  - $O(1)$ time for connected
  - $O(n)$ time for $n-1$ deletions
Decremental Connectivity in Trees
Decremental Connectivity in Trees

• Goal: Generalize two-level data structure from paths to trees.

• Simplifying assumption: Maximum degree of nodes in tree is 3.

• We need:
  • A balanced relabeling algorithm for trees.
  • An algorithm to divide trees into $O(n/w)$ subtrees of $\leq w$ nodes that only overlap in boundary nodes (does such division even exist?)
  • A fast data structure for decremental connectivity on subtrees with $\leq w$ nodes.
Balanced Relabeling for Trees
Balanced Relabeling for Trees

Replace left and right search with Breadth-first search away from edge
Balanced Relabeling for Trees

- Breadth-first search uses time linear in size of component.
- \(=\) Same analysis as with paths
- \(=\) \(O(n \log n)\) time for \(n-1\) deletes.
Summary

• Theorem: We can solve decremental connectivity in trees in
  • $O(1)$ time for connected.
  • $O(n \log n)$ time for $n-1$ deletes.
Tree Clustering
Tree Clustering

• Goal: Given a tree $T$ with maximum degree $\leq 3$ compute a cluster decomposition of $T$:
  • Divide $T$ into $O(n/w)$ connected subtrees ($clusters$) of $\leq w$ nodes.
  • Each cluster overlaps with other clusters in at most 2 boundary nodes.
Tree Clustering

• Lemma: A cluster decomposition exists and we can compute it in $O(n)$ time.

• Main idea:
  • Root $T$ at arbitrary node $r$.
  • Construct clusters greedily top-down.


For each child $c$ of root $r$:

- If $c$ has $\leq w - 1$ descendants. Form leaf cluster from $r$ and descendants of $c$ with $r$ as boundary node.

- If $c$ has $\geq w$ descendants. Pick descendant $z$ (possibly $c$ itself) of max depth to form internal cluster of $\leq w$ nodes with $r$ and $z$ as boundary nodes. Recurse on $z$. 

$w = 4$
Tree Clustering

• Why does the clustering algorithm produce $O(n/w)$ clusters each with $\leq w$ nodes?
• Each cluster has $\leq w$ nodes.
• How many clusters do we get?
• Intuition:
  • Greed $\Rightarrow$ A constant fraction of cluster will have $\Omega(w)$ nodes.
  • $\Rightarrow$ There will be a most $O(n/w)$ clusters.
Data Structure for Small Trees
Data Structure for Small Trees

• Theorem: We can solve decremental connectivity in trees with at most $w$ nodes in
  • $O(1)$ time for connected.
  • $O(1)$ time for delete
• Two-level data structure.

• Level 1: The $O(n/w)$ boundary nodes of the clusters. Edge between two boundary nodes iff connected in cluster. Maintain using balanced relabeling data structure.

• Level 2: Small tree data structure for each cluster.
• Analysis: Exactly as in the path case

• Theorem: We can solve decremental connectivity in trees in
  • O(1) time for connected
  • O(n) time for n-1 deletions
Summary

- Decremental Connectivity in Trees Problem
- Decremental Connectivity in Paths
  - First Tradeoffs
  - Two-Level Solution
- Decremental Connectivity in Trees
  - Two-Level Solution
References
