Centroid Path Handling Proof

October 15, 2018

Preliminaries

Define:

- $T[u]$ for some tree T and some node u to be the subtree of T rooted at u
- $\Lambda(T)$ for some tree T to be the leaf set of T
- Trees T_A and T_B on which the procedure `Filter_Clusters` will be run
- Centroid path of T_A composed of nodes $p_1 ... p_\alpha$, where p_1 is a leaf and p_α is the root

The portion of the `Filter_Clusters` algorithm that handles the centroid path is reproduced on the next page for ease.

We define a unit of work to be $O(\text{time taken to do a } BT \text{ operation}) = O(log\ n)$. Below, we show that we can assign every node a constant amount of work, thus resulting in a total time of $O(n\ log\ n)$.

We say a node is encountered if the algorithm reaches it, but does not change its membership of BT. We say a node is visited if the algorithm reaches it and changes its membership of BT.

Lemma 1: Any node is visited at most twice.
Proof: A node u is only removed from the BT when $counter(u) = |\Lambda(T_B[u]) \cap \Lambda(T_A[p_i])| = |\Lambda(T_B[u])|$ for some i. Thus, $\Lambda(T_B[u]) \subseteq \Lambda(T_A[p_i])$. Then u will never be added to BT again since Step 6 would never reach it (u is a descendant of r_i) and the loop in Step 10 would also never reach it (all leaves $x \in \Lambda(T_B[u])$ have already been handled). Note that this holds for steps $21-23$ also since $\Lambda(T_B[r_i])$ must be compatible with $\Lambda(T_A[p_i])$.

Lemma 2: For any leaf x, a maximum of 2 nodes are encountered during its treatment by this algorithm.
Proof: In each of the loops seen in Steps 11 and 17, nodes are added/removed from BT until a node causes the loop to terminate. All the nodes treated during the loops are *visited* since their status in BT changed. Thus only the last node treated, which caused the loop to terminate, is *encountered*.

Lemma 3: The remaining steps in the algorithm $(4, 5, 7-9, 24-28)$ take $O(n)$ time in total, so can be ignored.
Proof: Step 4 can be completed in $O(1)$ time. Step 5 can be completed in $O(|D|)$ time. Steps $24-28$ can be completed in $O(1)$ time. Total time over these steps $= \sum_{i=1}^{\alpha} O(1) + O(|D|) = O(n)$ since $\sum_{i=1}^{\alpha} |D| = n$

To conclude the analysis, note that during the execution of the algorithm, any node is only either encountered or visited. The total number of visitations $\leq 2 \times$ number of nodes $= O(n)$ [Lemma 1]. Also, total number of encounters $\leq 2 \times$ number of leaves $= O(n)$ [Lemma 2]. Thus total work assigned to any node is amortized $O(1)$. Since each unit of work done by a node is $O(log\ n)$, total time taken by the algorithm is $O(n\ log\ n)$.

```
Let $BT$ be an empty binary tree
Let $r_1$ := the leaf in $T_B$ labelled by $p_1$
 Set $counter(r_1) := 1$, and if $\alpha \geq 2$ then $counter(parent^{T_B}(r_1)) := 1$
for $i := 2$ to $\alpha$ do
    Let $D := \Lambda(T_A[p_i]) \setminus \Lambda(T_A[p_{i-1}])$
    Compute $r_i := lca^{T_B}(\{r_{i-1}\} \cup D)$
    Insert every node belonging to the path between $r_{i-1}$ and $r_i$ except $r_{i-1}$ into $BT$
    if $counter(r_{i-1}) < |\Lambda(T_B[r_{i-1}])|$ or $r_{i-1}$ is spoiled then
        insert $r_{i-1}$ into $BT$
    end
    for each $x \in D$ do
        while $x$ is not in $BT$ do
            Insert $x$ into $BT$
             $x := parent^{T_B}(x)$
        end
    end
    for each $x \in D$ do
        $counter(x) := counter(x) + 1$
        while $counter(x) = |\Lambda(T_B[x])|$ and $x$ is not spoiled and $x \neq root(T_B)$ do
            $counter(parent^{T_B}) := counter(parent^{T_B}) + |\Lambda(T_B[x])|$
             Remove $x$ from $BT$
             $x := parent^{T_B}(x)$
        end
    end
    if $r_i \in BT$ then
        remove $r_i$ from $BT$
    end
    if $BT$ is empty then $M := 0$;
    else $M$ := maximum weight of a node in $BT$;
    if $w(\Lambda(T_A[p_i])) > M$ then
        add $\Lambda(T_A[p_i]))$ to result tree with an $O(1)$ operation
    end
end
```