CS 345A Assignment 1

Yasharth Bajpai (170822), Nidhi Hegde (180472)

October 4, 2020

Difficult

Overview:

The algorithm devised follows a greedy strategy such that at each step the closest points in the graph are picked out and a smaller problem(A graph with one less point) is generated by following the procedure as described below.

Greedy Step:
The closest points are selected. Let these points be p_i, p_j where $d(p_i, p_j) = Min(d(p_x, p_y)) \forall p_x, p_y \in P$. Now these points are combined and replaced by a single compound point in the graph(P'). Let the new point generated be p_{ij}, such that $d(p_{ij}, p_k) = Min(d(p_i, p_k), d(p_j, p_k))$.
Thus, the graph P' is the smaller instance of the same problem. We recursively solve for the P' and obtain a solution $T(P')$. We then create a new tree node, make v_i & v_j its children and set the value of h as $d(p_i, p_j)$. We replace back the compound node in $P(T')$ by this new node.
The Greedy strategy is carried out recursively till the base case and every compound node in the solution of the smaller instance(that comes back up after recursion) is replaced by this newly created tree node to get the required solution for the current problem.

Base Case:
Once there are only two nodes left in the graph. If we go recurring beyond this, we would end up with a single point where the distance metric no more makes sense. Therefore, a new tree node is created and both these points are made it's children. The value h for this node is set equal to the distance between the two children. This tree node is returned as the solution of the current problem.

Pseudo Code:

Algorithm 1 Algorithm to produce a Hierarchical Metric τ

```
function HMetric(P)
    v_i, v_j, d_ij ⟵ FindMinEdge(P)
    if (n(P) == 2) then                          ▷ n(P) represents the number of points in P
        return ConstructNode(v_i, v_j, d_ij)
    end if
    P' = P \ {v_i, v_j} ⋃ v_ij
    N = ConstructNode(v_i, v_j, d_ij)
    T(P') = HMetric(P')
    T(P) = ReplaceNode(T(P), v_ij, N)
    return T(P)
end function
```

function ConstructNode(v_i,v_j,d_{ij})
  N =newTreeNode
  $left(N) \longleftarrow v_i$
  $right(N) \longleftarrow v_j$
  $parent(v_i) \longleftarrow N$
  $parent(v_j) \longleftarrow N$
  $h(N) \longleftarrow d_{ij}$
  return N
end function
function ReplaceNode($T(P')$,v_{ij},N) ▷ Replaces the compound node with the constructed node
  $parent(N) \longleftarrow parent(v_{ij})$
  if $(left(parent(v_{ij})) == v_{ij})$ **then**
    $left(parent(v_{ij})) \longleftarrow N$
  else
    $right(parent(v_{ij})) \longleftarrow N$
  end if
  $parent(v_{ij}) \longleftarrow Null$
  return $T(P')$
end function

Proof of Correctness:

Construction :

As stated in the question, we build a rooted tree T with n leaves, and we associate with each leaf node of T a point in the list P. For every point p_i in the list, its corresponding node in the tree is represented as a leaf node v_i.

Now, let p_i and p_j be the pair of points for which the distance function d is minimum.

Lemma1 : There exists an optimal full binary tree

Proof: If any node in the tree has just one child, then deleting that node wouldn't affect the optimality of the solution since the values for the rest of the nodes is left unchanged. Moreover, any node with just 1 child can never be a lowest common ancestor of any 2 nodes. Thus, we can perform this operation for any such node with one child without affecting the τ function's value for any 2 pair of points. Similarly, if any node n in the optimal solution has more than 2 children (let's say k children), then we can convert it into a binary tree by introducing $k-2$ new nodes each with the same height as that of n, as depicted in $Figure(1)$. We can guarantee that this won't affect the optimality of solution since the τ value of any 2 nodes whose LCA lie among the newly introduced nodes will be same as before, since the h value of their original parent is just copied down into these new nodes. Hence, we can construct a full binary tree out of a given optimal solution, while preserving the optimality of solution.

Lemma2 : There exists an optimal full binary tree in which the 2 nodes v_i and v_j are siblings

Proof: By the construction of full binary tree in $lemma1$, we can say that at the least, v_i must have a sibling. Let it be v_s. The LCA of v_i and v_s will be their immediate parent, and it will lie in the subtree rooted at the LCA of v_i and v_j. We can say that $\tau(v_i, v_j) \geq \tau(v_i, v_s)$. Since $\tau(v_i, v_j)$ is bounded above by $d(p_i, p_j)$ which is the minimum distance between any pair of points, all the heights in the subtree rooted at $LCA(v_i, v_j)$ must be bounded by it as well. Hence, if we swap the node v_s with v_j, the heights of all the

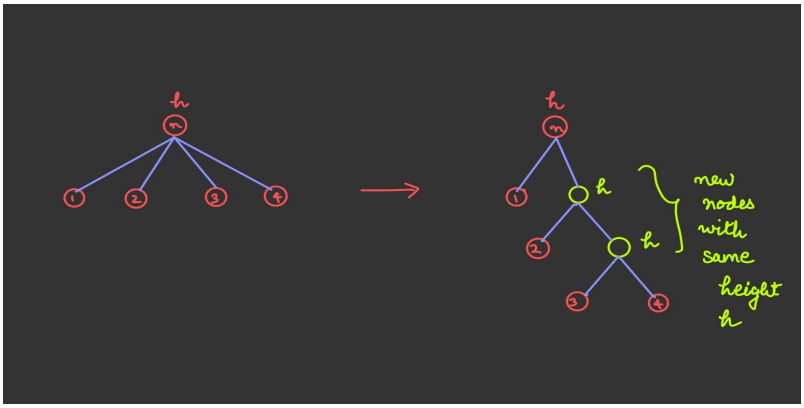

Figure 1: Case when a node has more than 2 children

nodes will still be consistent and need no change in their values. So, given an optimal full binary tree, we can translate it into an optimal solution where v_i and v_j are siblings.

Now, we may move on to finding a smaller instance of the problem. Let this instance be when the 2 sibling nodes v_i, v_j and their parent are replaced by a single special node v_{ij}. The corresponding distances of other points from this special node is defined as $d(p, p_{ij}) = min\{d(p_i, p), d(p_j, p)\} \quad \forall p \in P - \{p_i, p_j\}$.
Let's call these new set of points P' and the tree formed by them, $T(P')$. We need to estabilish a relation between $T(P')$ and the original problem $T(P)$.

We need to consider a property of these trees so that we can establish the relationship between them based on that property. So we define a function $S : T \to \mathbb{Z}^+$, such that it represents the sum of h values of all the nodes in a tree.

Step 1: Given an optimal solution for P' as $T_{opt}(P')$ we can construct a solution for P as $T(P)$, and establish a relation between $S(T(P))$ and $S(T_{opt}(P'))$
$T_{opt}(P')$ consists of $N-1$ leaf nodes by its definition. We can replace the compound leaf node by a tree rooted at a node n such that $h(n) \leq min(d(p_i, p_j), h(par))$, where par is the immediate parent of the leaf node being replaced. We impose this condition on $h(n)$ by the construction constraint of the tree given in the question. Thus, we now get a tree with N ($\because (N-1)-1+2$) leaf nodes such that the following relation holds:

$$S(T(P)) = S(T_{opt}(P')) + h(n)$$

Clearly, the value $h(par)$ can attain will be bounded by $Min_{\forall p \in P - \{p_i, p_j\}}(d(p, p_{ij}))$. Since the distance of compound node from any other node is strictly greater than $d(p_i, p_j)$, the optimal solution for $T(P')$ will have $h(par) > d(p_i, p_j)$. So we can set the height of the newly introduced node as $d(p_i, p_j)$ following the constraints defined in the problem. Hence, we get the following relation:

$$S(T_{opt}(P)) \geq S(T(P)) = S(T_{opt}(P')) + d(p_i, p_j) \tag{1}$$

Step 2: Given an optimal solution for P as $T_{opt}(P)$, we can derive a solution for P' as $T(P')$ and establish a relation between $S\left(T(P')\right)$ and $S\left(T_{opt}(P)\right)$

By *Lemma*2, there exists at least one $T_{opt}(P)$ in which v_i and v_j are siblings. Thus, if we replace the 3 nodes - v_i, v_j and their common parent v_n by a single special leaf node v_{ij} $[h(v_{ij}) = 0 \;\; by\; definition\; of\; T']$ and modify the distance function as $d(p, p_{ij}) \;=\; min\big\{d(p_i, p), d(p_j, p)\big\} \quad \forall p \in P - \{p_i, p_j\}$, we get a valid tree $T(P')$ consistent with the definition given in problem.

The following relation must hold between $T(P')$ and $T_{opt}(P)$:

$$S\left(T(P')\right) = S\left(T_{opt}(P)\right) - h(n)$$

Again since the node removed is the parent of v_i and v_j, its height is bounded by $d(p_i, p_j)$, with equality holding in the case of optimal solution for $T(P)$. Thus, we get:

$$S\left(T_{opt}(P')\right) \geq S\left(T(P')\right) = S\left(T_{opt}(P)\right) - d(p_i, p_j) \tag{2}$$

Therefore, using (1) & (2) an equality holds when $T(P)$ & $T(P')$ are both optimal, implying that an optimal solution for P' corresponds to an optimal solution for P.

Time Complexity:

At every step, only a single recursion to a smaller problem is called. The depth of the recursion stack is bounded by n(The number of points in the graph) with Polynomial time manipulations at each level. Thus, the algorithm has a polynomial bound.

To be precise, if we employ a suitable structure such as a MinHeap for storing & returning the minimum edges. The complexity of each step becomes $O(n \log n)$. Thus, the overall complexity turns out to be $O(n^2 \log n)$.