AMDM-3

Abhilash Jain

November 2018

1 Clustering given a hierarchy

1.1 Algorithm that can be solved in polynomial time

Given : We are given a set X of n points, a hierarchy binary tree T over X, as described above, and an integer k. We want to find k nodes $(v_1, ..., v_k)$ of T, such that $\bigcup_{i=1}^{k} X(v_i) = X and X(v_i) \cap X(v_j) =$ Not Zero for all i $\neq$ j and such that the objective function, $\sum_{i=1}^{k} var(v_i)$ is minimized
where var(v$_i$)$\sum_{x \in X(v_i)} ||x - c(v_i)||^2$ where c(v) is the mean of all points in X(v).

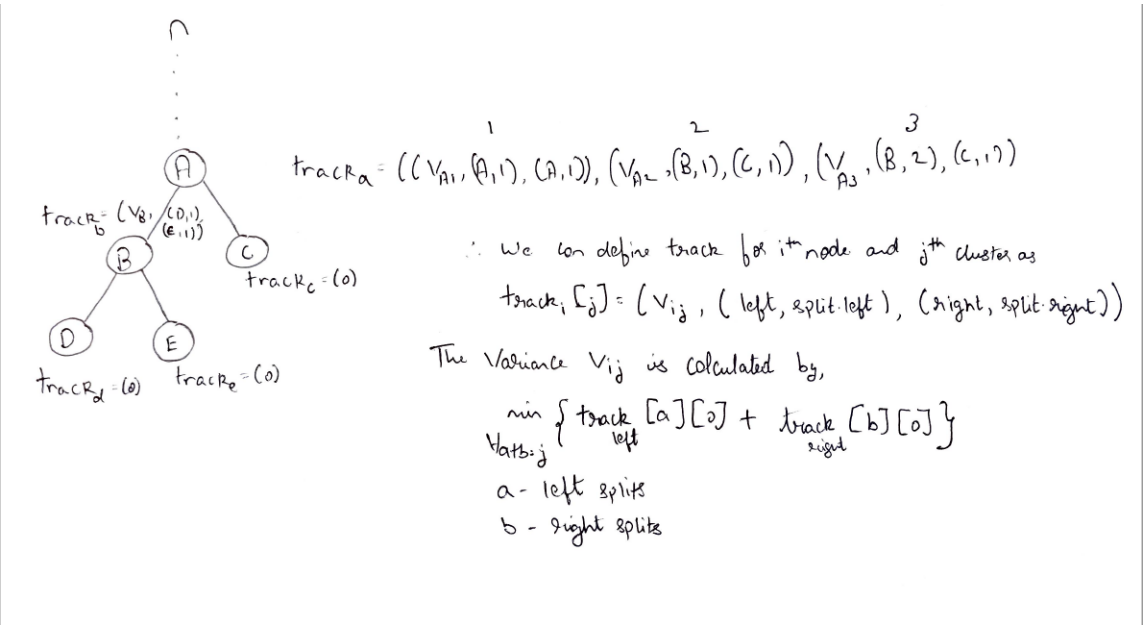
Solution: The problem can be solved optimally in polynomial time, and below is first the explanation of such said algorithm and then the algorithm.
Apart from the question, lets also have the following definitions for a node as well,i.e define some structure to a Node:

- Left, right - the left and right child of the node
- value - value of the Node
- var - The variance of the Set at that node calculated by the formula given in the question
- mean - The mean of the Set at that node
- track$_i[j]$ - A Tuple track which is very important and and has the functionality, that, for each corresponding index we will have the variance if it was part of those many clusters and the breakdown of the clusters, where i is the node and j is the number of clusters we want for that node.

There-fore a Tree variable 'root' will have the entire Tree structure and Data.

The Algorithm and How it works:
The main motivation behind the Algorithm is that the Variance of the parent node is the combined variance of the child nodes(clearly from the problem), which signifies that we can track how we arrived at this variance recursively and there-fore keep a map of how we can minimize this variance for the clusters

we require(as this information is available to us)
This map is nothing but our tuple track, Below is the illustration of how we came up with this track



Lets consider a part of Tree as described above which can go upto 'n' nodes, for all the leaf nodes, our Variance will be ZERO and they cannot be broken down into clusters. Therefore, $\text{Track}_{leaf} = 0$
and as we keep going up levels of the tree, our tuple for i^{th} node keeps tracking the Combined variance and cluster splits for the j^{th} cluster. The Mean and The Combined Variance [1] of the parent can be calculated as

$$Mean_{Parent} = \frac{N_{left} \cdot \mu_{left} + N_{right} \cdot \mu_{right}}{n_{left} + n_{right}} \tag{1}$$

$$Variance_{Parent} = Variance_{Left} + Variance_{Right} + N_{left} \cdot (\mu_{Left} - \mu_{Parent})^2 + N_{Right} \cdot (\mu_{Right} - \mu_{Parent})^2 \tag{2}$$

where N is the Number of elements

This approach is very similar to Dynamic Programming and such Table could also be implemented but i did not choose it as there would be wastage of space in the table.

THE ALGORITHM:
The Algorithm is split into two:

- Process_Track Function: We perform a Post-Order Traversal and start from the left most leaf node and start calculating the variance and the Track tuple of each node.

- Get_Clustering: We Define a function to return the optimal clusters on the Tree which has been processed through the above function.

Algorithm 1: Process_Track

```
Data: root
begin
    if root ≠ Null: then
        Process_Track(root.left);
        Process_Track(root.right);
        if Root is leaf node: then
            Set root.mean as root.val;
            Set root.var as 0;
            Set root.count as 1;
            Set Length root.Track as root.Count;
            Set root.Track[1] Variance as root.var;
        else
            Set root.mean using equation (1);
            Set root.var using equation (2);
            Set root.count as root.left.count + root.right.count;
            Set Length of Track as count;
            Set Track[1] Variance as root.var;
        end
        for i in (length of left side) do
            for j in (length of right side) do
                Minimizing Variance of Track_i[j];
                Set Track_i[i+j][0] as Variance;
                Set Track_i[i+j][1] as i;
                Set Track_i[i+j][2] as j;
            end
        end
    else
end
```

After we Process the Binary Tree we define a Get_Clustering function to get the optimal Clustering

Algorithm 2: Get_Clustering

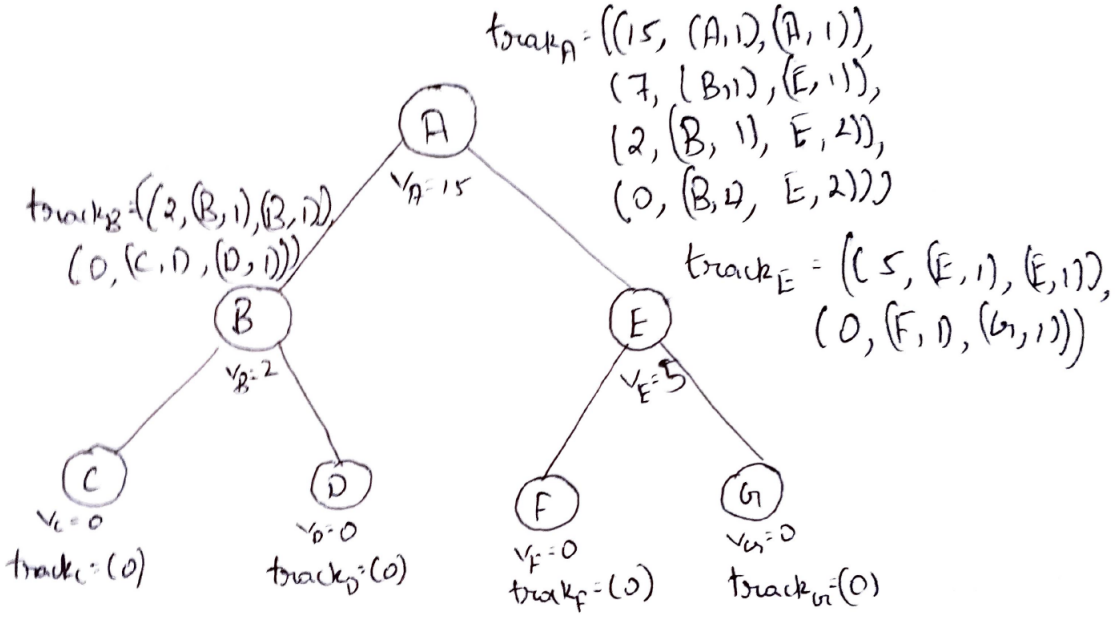
```
Data: root,NoOfClusters
Result: Clusters: Set of Nodes which are optimal Clusters
1 begin
2     if Track_i reaches 1 cluster then
3         Clusters += root.value;
4     else
5         Get_Clustering(root.left,Track_i[NoOfClusters][1] or
            root.right,Track_i[NoOfClusters][2]);
6     end
7 end
```

1.2 Correctness of the Algorithm

The Premise of the Algorithm is two fold:

- The Variance of the Parent Node is the Combined Variance of the Child Node, signifying that the Parent 'Knows' it's children, thus forming a Tree where in the root can 'know' all the variances of the nodes
- We are also Tracking the best possible splits or clusters which can occur at the node i.e. choosing the most optimal clusters by minimizing the variance (Meaning, we chose the split which has the least variance)

Due to the above approach,Each node has the most optimal cluster information, and this gets passed on to their parents, We are performing a Post Order Traversal there-fore we know that, if The Child has the most optimal Solution, and as we go Up traversing, The parent node will have the optimal Solution.Therefore the Root will have the optimal Solution for the Binary Tree, Consider the Following Example,



In this Example, lets consider the Following Cases:

- NoOfClusters=1, it would just Return the Root Node, with Variance 15
- NoOfClusters=2, it would Return Two Clusters,B and E with Variance 7
- NoOfClusters=3, it would Return 3 clusters, B and Split E into F and G with Variance 2
- NoOfCluster=4, it would return the 4 leaf Nodes by splitting B and E

We were able to deduce that from the root we can form the number of clusters required, and due to the relationship between the parent and child, we know exactly who to split and how, Ipso Facto, The Algorithm reproduces the optimal number of Clusters by minimizing the Object Function in the Question.

1.3 Complexity of the Algorithm

In the Following Algorithm, A tree with 'N' nodes,For the worst case our performance for each snippet of the Algorithm would be:

- A Post Order Traversal $\mathcal{O}(n)$
- Two For Loops to calculate the Track Tuple $\frac{n}{2}$ for each left and right side,i.e. $\mathcal{O}(\frac{n}{2} \cdot \frac{n}{2}) = \mathcal{O}(\frac{n^2}{4}) = \mathcal{O}(n^2)$
- To generate the Clusters $\mathcal{O}(n)$

There-fore the Total Complexity is $\mathcal{O}(n^2 \cdot n + n) = \mathcal{O}(n^3)$

2 Clustering a Data stream

2.1 Clustering that has at most k cluster centers

Motivation : The motivation mainly for this proof comes from the slides and the definitions in the problem. And we would prove that STREAMING-FURTHEST algorithm produces a clustering that has at most k cluster centers by: **Proof of Contradiction**.

Let's Consider a Case were we have found our 'K' optimal clusters, but now 'K+1' clusters is possible, that would mean we have some new point p which would make it possible, Now by definition of the STREAMING-FURTHEST algorithm,

$$d_m(p, x) > 2d^*$$

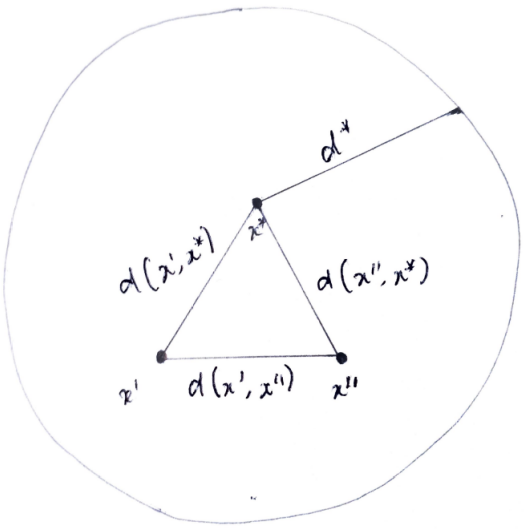
Where, x is the closest cluster center in C There-fore, ever pair of centers in this new 'K+1' Clusters, by definition, would have the following:

$$d(x^{'}, x^{"}) \geq d_m(K + 1)$$

Which means that,

$$d(x^{'}, x^{"}) > 2d^*$$

Now, we know that the STREAMING-FURTHEST Algorithm computes 'K' clusters using the optimal d^*, and Hence the K+1 clusters which we are considering must all fall into one of the K optimal cluster, By Pigeon Hole Principle there must be one optimal centre x^* which contains two centres $x^{'}$ and $x^{"}$ from the set of K+1 centres,and can be represented something like this:



There-fore by Triangular Inequality for $x^*, x^{'}, x^{"}$, we arrive at:

$$d(x^{'}, x^{"}) \leq d(x^{'}, x^*) + d(x^{"}, x^*)$$

And we know that within a cluster, the distance between the Centre and any point has to be $\leq d^*$. There-fore:

$$d(x^{'}, x^*) \leq d^*$$

$$d(x^{"}, x^*) \leq d^*$$

There-fore, substituting above:

$$d(x^{'}, x^{"}) \leq d(x^{'}, x^*) + d(x^{"}, x^*) \implies \leq 2d^*$$

$$\implies d(x^{'}, x^{"}) \leq 2d^*$$

There-fore we confronted with a contradiction,
Hence, STREAMING-FURTHEST algorithm produces a clustering that has at most k cluster center.

2.2 Algorithm is still a factor-2 approximation

The proof is very similar to the proof of 2.1 and the premise of both of them remain the same(ipso facto), We can try to prove this by proof of contradiction. We know that STREAMING-FURTHEST Algorithm gives us set C of cluster centers with $2d^*$.
lets consider from the algorithm we output a set C+1 clusters with $2d^*$
But from the definitions in the problem, the optimal solution of cost d^* for set C of Clusters is known, There-fore the arriving at a solution for C+1 clusters is contradicting to the definition in the problem and d^* is not optimal.
Because of the contradiction, we can safely conclude that the STREAMING-FURTHEST algorithm is still a factor-2 approximation of the optimal clustering C^* on the data stream X.

The same figure as 2.1 is relevant in this context as well

2.3 Modifying the STREAMING-FURTHEST algorithm

In the case where-in we do not know the optimal d^*, we can try to approximate it using the cost calculation formulae from the slides at each iteration.

$$\max_{i=1}^{n} \min_{j=1}^{k} ||x_i - c_j||_2$$

We first treat the first 'k' data points as k centres, and keep inserting them in $X^{'}$ and the cluster centres in $C^{'}$. From 'k+1' data points we start minimizing the cost and iteratively keep updating the Cluster centres $C^{'}$, as $X^{'}$ keeps increasing.You can think of updating cluster as follows as this would be more important:When-ever a new data point d' arrives with its distance greater than the cost from any cluster centre, we replace this d' with a Supposed cluster centre $c^{'}$ from $C^{'}$,closest to it.

3 Connectivity of a Graph

3.1 Streaming Algorithm for deciding the connectivity of G(T,W)

Motivation: The motivation of the Algorithm is based on the following:

- To reduce the traversals on the edges of our Graph, we need to somehow reduce the redundant edges, We use an adaptation of Minimum Spanning Tree, in the sense that, when performing this operation,if a cycle is found we remove the oldest edge in that cycle.(Motivated by [2] and [3]). We define this function as MinGraph.
- We define a Queue function where length of queue is the window length and this takes care of removing the oldest element and adding the newest element. We define this function as Queue_w, where w is the length of the sliding window

We also maintain the information of edges the same way defined in the problem e_i, where i is the i^{th} position in Time T
and e_i.Time lets us know this T We know that for a Minimum Spamming Tree can at-most have $V-1$ edges, where V is the number of Edges, Therefore for the graph to be connected, we check the condition $V-1$

Algorithm 3: Connectivity

```
Data: Graph G with edges e //Stream of Edges, V-Number of Vertices
Result: At each iteration 1: if connected, 0: if not connected
1  begin
2      for i in (length(Stream)) do
3          Set G' with new Incoming Edge e_i
4          Set MinG with the output of MinGraph(G') //We have the
             Minimum Spanning Tree in MinG
5          Set G_a after passing it through Queue_w(MinG,i) //The Queue
             Function acts like Window
6          if |G_a| = |V - 1| then
7              Print 1 //Graph is Connected
8          else
9              Print 0 //Graph is Disconnected
10         end
11     end
12 end
```

```
Function Queue_w(MinG, i):
    foreach e_i in MinG do
        if e_i.Time > i − W then
            Append G with e_i
        else
    end
    Return G
```

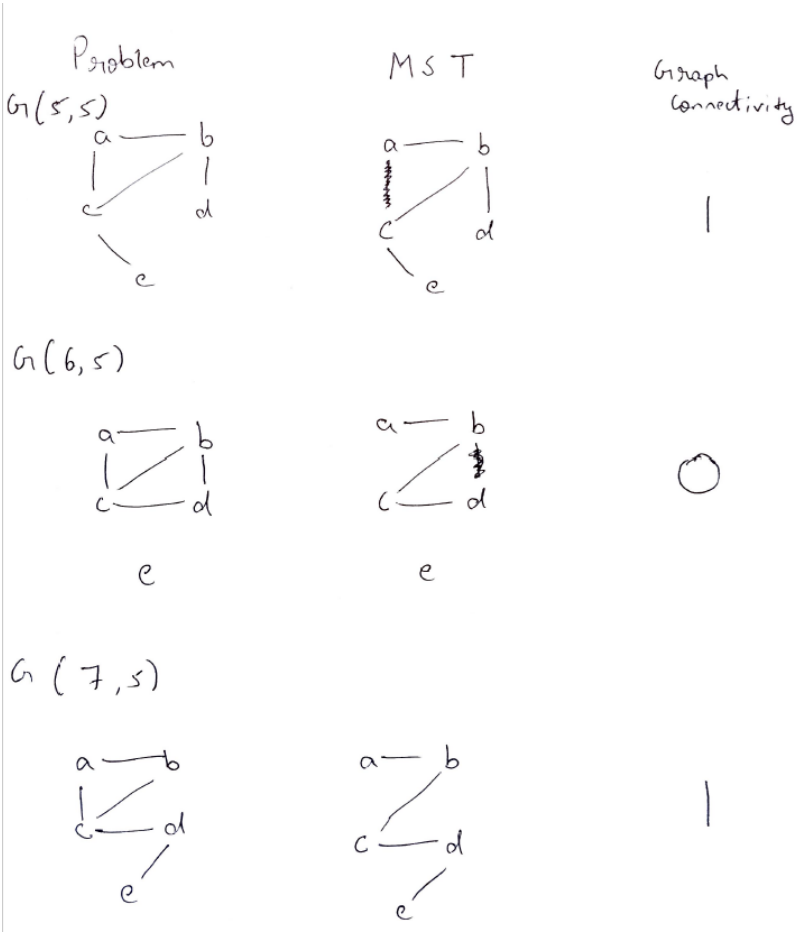
3.2 Correctness of the Algorithm

There are Two Main aspects to the Algorithm:

- The Minimum Spamming Tree of the graph represents the Structure of the graph, and by definition a minimum Spamming Tree only can be used to decide whether the graph is connected or not. And in our Algorithm we remove the oldest edge in the cycle, as it benefits us anyways as we are going to remove old edges.
- The Queue Function maintains the sliding window functionality, In principle, Sliding Window functions like a Queue, with 'First In First Out' Logic. For our Function,Let's take Edges e_i at any Time T for Window length W, The condition, will not append e_i if it is smaller than $T - W$, as given in the question. and if $T \leq W$. We keep all the edges till $T > W$. Which is how the question wants us to do it.

We then determine the connectivity of the graph as 1, if $G(T, W) = N - 1$ else 0, $N - 1$ due to the fact that a Minimum Spanning Tree can have at most $N - 1$ edges, if N is the number of Vertices.

Consider the Example given in the Problem:



- G(5,5) : We have removed the edge (a,c), and it still doesn't affect our outcome to determine the connectivity, We can still determine it to be 1
- G(6,5): We removed the edge (b,d), and same as above we are able to determine the connectivity. More-so, We removed the edge (b,d), which was anyways going to be removed at G(7,5)

In Conclusion, The Algorithm, not only works as expected, but does so optimally.

3.3 Space of the Algorithm

We Reuire the space for the following:

- Space for the MinG graph.i.e the Minimum Spamming Tree, which can at most be N-1 edges .i.e $\mathcal{O}(N-1) = \mathcal{O}(N)$
- In the Queue function to save, again at most will be N-1 edges as we process the MinG graph .i.e $\mathcal{O}(N)$ (Even if Window Size is the same size as the Stream)

There-fore, even when we combine them the space complexity at most will be $\mathcal{O}(N)$

3.4 Update time of your Algorithm

The function time which will consume the most time will be the MinGraph function. The minimum Spamming tree updation of N Vertices and N-1 Edges, will take $\mathcal{O}(N + |N-1|) = \mathcal{O}(N)$

4 Acknowledgements

Following are some refrences I have used:
[1] - https://www.emathzone.com/tutorials/basic-statistics/combined-variance.html
[2] -https://people.cs.umass.edu/ mcgregor/papers/13-graphsurvey.pdf
[3] -https://people.cs.umass.edu/ mcgregor/papers/13-esa.pdf
Below are the specific people acknowledgements:

- Discussed with the Macademia group in a meeting hall about general approaches for all the problems
- **Ananth Mahadevan, ChristaBella, Alexandru** - Regular discussions, sharing of opinions and ideas on all the problem approaches took place. (where else but Paniikki)