CS-E4600 — Homework #3

fall 2018

Due: Fri, Nov 9, midnight

Return your answers as a PDF file via `mycourses.aalto.fi`

Discussing the problems with your colleagues is allowed. However, you should write the answers by yourself. If you discuss with others, give proper acknowledgments. Looking for the answers in the internet is discouraged. You should at least make a serious effort to solve a problem by yourself before looking online. If you do, however, give proper acknowledgments.

Remember that there is a budget of 5 late days, which you can use in any way you want for the three homeworks and the programming assignment. Weekend days count as late days.

Typed solutions are strongly encouraged, especially if your hand writing is messy.

"I do not know" answers receive 15% of the max score for each problem. Partial credit will be given for partial solutions, but long off-topic discussion that leads nowhere may receive 0 points. Overall, think before you write, and try to give concise and crisp answers.

Problem 1 : clustering given a hierarchy [30 points]

Consider again a set X of n points in $\mathbb{R}^d$. In addition, this time, we are given a hierarchy tree T over the points of X.

The tree T is binary and each node $v \in T$ corresponds to a subset of X. Let us denote by $X(v)$ the set of points that correspond to node v. There are exactly n leaves in T, each one corresponding to exactly one point in X. If v, u, w are nodes of T with u and w being the children of v then $X(v) = X(u) \cup X(w)$. Thus, the root r of the tree corresponds to the whole set X, i.e., $X(r) = X$.

We want to solve the k-means problem on X, but the resulting clustering is constrained by the nodes of the tree. More precisely, we want to solve the following problem.

k-means on trees: We are given a set X of n points in $\mathbb{R}^d$, a hierarchy binary tree T over X, as described above, and an integer k. We want to find k nodes $\{v_1, \ldots, v_k\}$ of T, such that $\bigcup_{i=1}^{k} X(v_i) = X$ and $X(v_i) \cap X(v_j) = \emptyset$ for all $i \neq j$, and such that the objective function

$$\sum_{i=1}^{k} \mathrm{var}(v_i)$$

is minimized, where $\text{var}(v_i) = \sum_{x \in X(v_i)} ||x - c(v_i)||^2$ and where $c(v)$ is the mean of all points in $X(v)$.

Question 1.1 [15 points] Show that the problem of k-means on trees can be solved optimally in polynomial time.

Question 1.2 [10 points] Show the correctness of your algorithm.

Question 1.3 [5 points] What is the complexity of your algorithm?

Problem 2 : clustering a data stream [35 points]

Recall the k-centers problem:

[k-centers] : Given a set X of n points in $\mathbb{R}^d$ and an integer k, find k points $\{p_1, \ldots, p_k\}$ in $\mathbb{R}^d$ (not necessarily in X) so that the clustering objective

$$\max_{x \in X} \min_{i=1}^{k} ||x - p_i||$$

is minimized.

Recall the furthest-first traversal algorithm (by Gonzalez, published in 1985) designed for the k-centers problem. Let us refer to this algorithm as FURTHEST.

Algorithm FURTHEST
Input: set of points X, number k
Output: clustering of points in X in k clusters
1. pick any data point in X and label it x_1
2. **for** $i = 2, ..., k$
3. find the unlabeled point that is furthest from $\{x_1, x_2, ..., x_{i-1}\}$
4. label that point x_i
5. assign the remaining unlabeled data points to the closest labeled data point
6. **return** the labeled points (cluster centers), and the assignment of each unlabeled point to its closest cluster center

As mentioned in the class the FURTHEST algorithm provides a 2-approximation to the k-centers problem. Namely let C^* be an optimal solution to the k-centers problem and d^* be its cost (maximum radius in C^*). Let also C_F be the solution found by the FURTHEST algorithm and let d_F be its cost (maximum radius in C_F). Then

$$d_F \leq 2\, d^*.$$

A sketch of the proof is given in the course slides. Your first task is to study the proof and make sure that you understand it. If you have any questions please ask the instructor or the TAs.

We want to adapt the FURTHEST algorithm so that it works in a *streaming* setting: the data points in X arrive in a streaming fashion, one after the other, and we want to compute a clustering of the data points in X in a single pass.

Assume first that the cost d^* of the optimal clustering C^* of k-centers in X is *known*. In this case, we can make a streaming version of the FURTHEST algorithm that works as follows.

Algorithm STREAMING-FURTHEST
Input: a stream of data points X
(Assume that the optimal k-centers cost d_* is known)
Output: clustering of points in X in k clusters
1. read x_1, the first point in the stream
2. define the set of cluster centers as $C \leftarrow \{x_1\}$
3. **do** read the next point x in the stream
4. compute the distance d_m from x to its closest cluster center x_i in C
5. **if** $d_m > 2\,d^*$
6. take x to be a new cluster center
7. $C \leftarrow C \cup \{x\}$
8. **else** assign x to its closest cluster center (which is x_i)
9. **until** the data stream finishes
10. **return** the set C of cluster centers

Question 2.1 [15 points] Prove that the STREAMING-FURTHEST algorithm produces a clustering that has at most k cluster centers.

Question 2.2 [10 points] Prove that the STREAMING-FURTHEST algorithm is still a factor-2 approximation of the optimal clustering C^* on the data stream X.

Question 2.3 [10 points] Suggest how to modify the STREAMING-FURTHEST algorithm for the (realistic) case that the cost d^* of the optimal clustering is not known.

Hint: You may want to answer **2.2** and **2.3** even if you have not answered **2.1**.

Problem 3 [35 points]

We are monitoring a graph, which arrives as a stream of edges $\mathscr{E} = e_1, e_2, \ldots$. We assume that exactly one edge arrives at a time, with edge e_i arriving at time i, and the stream is starting at time 1. Each edge e_i is a pair of vertices (u_i, v_i), and we use V to denote the set of all vertices that we have seen so far.

We assume that we are working in the "sliding window" model. According to this model, at each time T only the W most recent edges are considered *active*. Thus, the set of active edges $E(T, W)$ at time T and for window length W is

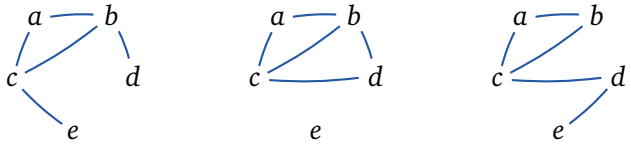
$$E(T,W) = \begin{cases} e_{T-W+1}, \ldots, e_T, & \text{if } T > W, \\ e_1, \ldots, e_T, & \text{if } T \leq W. \end{cases}$$

We then write $G(T, W) = (V, E(T, W))$ to denote the graph that consists of the active edges at time T, given a window length W.

As an example, given the stream of edges

$$e_1 = (c, e),\ e_2 = (b, d),\ e_3 = (a, c),\ e_4 = (c, b),\ e_5 = (a, b),\ e_6 = (c, d),\ e_7 = (d, e)$$

the graphs $G(5,5)$, $G(6,5)$ and $G(7,5)$ are shown below:



$G(5,5)$ $G(6,5)$ $G(7,5)$

Notice that for all $T \geq W$ the graph $G(T+1, W)$ results from $G(T, W)$ by adding one edge and deleting one edge.

We want to monitor the connectivity of the graph $G(T, W)$. In other words, we want to design an algorithm that quickly decides, at any time T, if the graph $G(T, W)$ is connected. In the previous example, the graphs $G(5,5)$ and $G(7,5)$ are connected, while the graph $G(6,5)$ is not connected.

Question 3.1 [15 points] Propose a streaming algorithm for deciding the connectivity of $G(T, W)$.

Hint for **3.1***:* An efficient streaming algorithm takes advantage of the fact that the graph $G(T+1, W)$ changes very little compared to $G(T, W)$. Therefore, our algorithm should be able to efficiently update the connectivity of $G(T+1, W)$ when a new edge e_{T+1} arrives at time $T+1$, given that the connectivity of $G(T, W)$ has already been computed.

Question 3.2 [10 points] Prove the correctness of your algorithm.

Question 3.3 [5 points] How much space does your algorithm use?

Question 3.4 [5 points] What is the update time of your algorithm?

Hint for **3.3** *and* **3.4***:* The space of your algorithm is the maximum amount of space used at any given moment. The update time is the time needed to compute the output at time $T+1$, given the state at time T and the new edge e_{T+1} that arrives at time $T+1$.

You should provide your answer using the $\mathcal{O}$ notation, written as a function of the window length W and the number of vertices $N = |V|$.