General rules for solving exercises

- This is a theory course, which means: if an exercise does not explicitly say “you do not need to prove your answer” or “justify intuitively”, then a formal proof is always required.
- All exercises and their solutions, no matter whether they are graded or regular/optional ones, are part of the material relevant for the two exams.
- Some of the exercises are marked as “in-class”, which means that we do not expect you to solve them before the exercise session. Instead, your teaching assistant will solve them with you in class.
- You are highly encouraged to solve all other exercises (those not marked as “in-class”) on your own and to hand in a writeup of your solutions no later than the due date. If you choose to do so, please write the name of your teaching assistant on the front sheet.

The following exercises will be discussed in the exercise class on October 15, 2014. Since we expect you to be working on the first special assignment, all exercises are in-class and there is no due date.

Exercise 1: Farthest Point Voronoi Diagrams (in-class)

You want to open a restaurant – let us call it “Rose’s”– that will have its food delivered to customers by bike messengers. This is a somewhat demanding endeavor, as you need to avoid the worst case that the food gets cold before it reaches the customer; i.e., you want to keep an eye on the largest distance to a customer. You already have the coordinates of your most important future customers. In order to make an informed decision on where to rent a place for your business, you want to draw a map that lists for all places in the city the farthest future customer. Assume you do not live in Zurich but in a city where Euclidean distances are a valid approximation of biking times.

As you certainly already have guessed, a locus approach is advisable here. We abstract the customers to a set \( S \) of \( n \) points in the plane, out of which no three lie on a common line and no four on the same circle. The structure we are looking for goes by the name of farthest point Voronoi diagram. It divides the plane into regions that have the same farthest point in \( S \).

(a) Express \( V^F_S(p) \), the farthest point Voronoi cell of \( p \), i.e. the set of points in the plane for which \( p \) is the farthest point in \( S \), in terms of \( h(p, p') \) as defined in the lecture notes on page 53 (Chapter 2).

(b) Unlike \( V_S(p) \), \( V^F_S(p) \) does not necessarily contain a point. Give a small example \( S_b \) of a point set and a point \( p \in S_b \) for which \( V^F_{S_b}(p) \) is empty.
Exercise 2: Linear Separability in Linear Time (in-class)

Suppose we are given a set $S$ of $n$ closed halfspaces in the plane. For each $H \in S$, let $\ell_H \subset H$ denote its boundary line. We assume that the halfspaces are in general position such that no two boundary lines are parallel and no three boundary lines meet in a single point. Consider the input to be given in the form of linear inequalities, say.

In this task we are interested in a randomized algorithm to decide whether the intersection of the given halfspaces is non-empty, that is whether $R(S) = \emptyset$ for $R(S) := \bigcap_{H \in S} H$, or not. If $S$ has a non-empty intersection, we would also be interested in a certificate point, that is in a point $x \in \bigcap_{H \in S} H$ to demonstrate non-emptiness. To make your calculations simpler, we want to make certificate points unique. To this end, we assume $|S| \geq 2$ and fix, arbitrarily, two halfspaces $H_1, H_2, \in S$. The region $R(S)$ is obviously contained in a wedge formed by the lines $\ell_{H_1}$ and $\ell_{H_2}$ (see figure). Before starting any algorithm, you may assume that the input is rotated so first in such a way that this wedge opens to the right and the intersection point $g \in \ell_{H_1} \cap \ell_{H_2}$ acts as a guard that no point in $R(S)$ can have a smaller $x$-coordinate than $g$ (see figure). We then define for any $S' \subseteq S$ with $H_1, H_2 \in S'$ the unique certificate point $c(S')$ as the point in $R(S')$ that has the smallest $x$-coordinate. You may assume that $H_1$ and $H_2$ are fixed before and known to all your algorithms below.

Following are your tasks:

(a) Let $|S| \geq 3$ (with $H_1$ and $H_2$ as described above) and let $H \in S \setminus \{H_1, H_2\}$ be an arbitrary one of the halfspaces. Prove: if $R(S) \neq \emptyset$, then either $c(S) = c(S \setminus \{H_1\})$ or $c(S) \in \ell_H$.

(b) Let $|S| \geq 3$ (with $H_1$ and $H_2$ as described above) and let $H \in S \setminus \{H_1, H_2\}$ be an arbitrary one of the halfspaces. Assume that $R(S \setminus \{H_1\}) \neq \emptyset$. Write down a deterministic algorithm that runs in time linear in $n = |S|$ and that on input $(S, H, c(S \setminus \{H_1\}))$ determines whether $R(S) \neq \emptyset$ and if so outputs $c(S)$.

---

1 this rotation can always be done such that we also do not have vertical or horizontal lines, which you may assume
(c) Let again \(|S| \geq 3\) (with \(H_1\) and \(H_2\) as described above). Using (b), write down a randomized algorithm which, given \(S\), determines whether \(R(S) \neq \emptyset\) and if so outputs \(c(S)\). Your algorithm should run in expected time linear in \(n = |S|\).

Finally, let us conclude with a beautiful application:

(d) Consider the following fundamental problem for classification methods. Given a set of red and blue points in general position in the plane. General position here means that no three points lie on a common line. Find a line separating the two point classes (if such a line exists).

Show that this problem can be solved in expected linear time (in the total number of points) by a randomized algorithm.

Exercise 3: Approximating the Minimum Cut (in-class)

(Exercise 3.3 from the lecture notes)

You recall that the algorithm \textsc{BasicMinCut} computes a guess for the size of a minimum cut of a (multi)graph \(G\) by repeatedly contracting a uniformly random edge until there are only two vertices left and then returning the number of edges running between these two vertices.

As usual, denote the size of a minimum cut of \(G\) by \(\mu(G)\). We have derived in the lecture that the number \(L_G\) which \textsc{BasicMinCut} outputs (on input \(G\)) is at least \(\mu(G)\), and \(\Pr[L_G = \mu(G)] = \Omega(n^{-2})\).

Consider the following slightly modified algorithm \textsc{BasicMinCut'}: just like \textsc{BasicMinCut}, it repeatedly contracts a uniformly random edge until there are only two vertices left. But instead of just returning the number of edges between those two vertices in the very end, it returns the smallest degree of any vertex observed during the execution of the algorithm. That is, if \(G_0, G_1, G_2, \ldots, G_{n-2}\) is the sequence of graphs encountered, with \(G_0 = G\) and \(|V(G_{n-2})| = 2\), it returns

\[ L_G := \min_{0 \leq i \leq n-2} \min_{v \in V(G_i)} \deg(v). \]

Prove that

(a) \textsc{BasicMinCut'} can be implemented so as to run in time \(O(n^2)\),

(b) \(L_G \geq \mu(G)\) always holds,

(c) for any fixed \(\alpha > 0\), the success probability

\[ p_\alpha(n) := \min_{G \text{ a graph on } n \text{ vertices}} \Pr[L_G \leq (1 + \alpha)\mu(G)] \]

satisfies the recurrence

\[ p_\alpha(n) \geq \left(1 - \frac{2}{(1 + \alpha)n}\right) p_\alpha(n-1). \]

Using (c), one can prove that for any fixed \(\alpha > 0\), \(p_\alpha(n) \in \Omega(n^{\frac{1}{1+\alpha}})\), but this is just calculation and we do not ask you to do this here.

3