Exercise Set XI

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

These problems are taken from various sources at EPFL and on the Internet, too numerous to cite individually.

1 Consider two LSH hash families $\mathcal{H}_1$ and $\mathcal{H}_2$ designed for a distance function $\text{dist} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. For $r = 0.1$ and $c = 2$, $\mathcal{H}_1$ satisfies

$$\text{dist}(p,q) \leqslant r \implies \Pr_{h\sim\mathcal{H}_1}[h(p) = h(q)] \geqslant 1/2$$
$$\text{dist}(p,q) \geqslant c \cdot r \implies \Pr_{h\sim\mathcal{H}_1}[h(p) = h(q)] \leqslant 1/8$$

and $\mathcal{H}_2$ satisfies

$$\text{dist}(p,q) \leqslant r \implies \Pr_{h\sim\mathcal{H}_2}[h(p) = h(q)] \geqslant 1/8$$
$$\text{dist}(p,q) \geqslant c \cdot r \implies \Pr_{h\sim\mathcal{H}_2}[h(p) = h(q)] \leqslant 1/200$$

1a Which Hash family would you choose to build the data structure ANNS(r, c) explained in class? What would the space requirement and query time be (logs are not so important)?

1b On query $q \in \mathbb{R}^d$, asymptotically how many hash function computations are done?

Solution: 1a: When building the data structure ANNS(c, r) from an $(r, c \cdot r, p_1, p_2)$-LSH family, the key parameter is ρ in the equation $p_1 = p_2^\rho$. Basically, small ρ yield good runtimes and space requirements (note that ANNS assumes $p_1 > p_2$, so $0 < \rho < 1$). The first family has $\rho = 1/3$ and the second $\rho \approx 0.39$, so the first family is the most suited for ANNS.

Let us explore the space requirements of ANNS with $\rho = 1/3$. To ease the notation we set $|P| = n$. The preprocessing step creates ℓ hash tables where each table maps the search space P to some k-dimensional vectors. The space requirement is therefore $O(\ell \cdot n \cdot k)$. Now, to output a correct response with probability at least $1-1/n$, we need to have $k \in O(\ln n)$ and $\ell \in O(n^\rho \ln n)$. This yields a space complexity of order

$$O\left(n^{4/3} \ln(n)^2\right)$$

To derive the query time, let us assume that computing $dist(p, q)$ for some $p, q \in \mathbb{R}^d$ takes time $O(d)$ (for instance, this is the complexity of computing the standard euclidean distance). At query time for input $q \in \mathbb{R}^d$, we need to iterate over the ℓ hash tables, each time computing the hash of q and looking for collisions. Computing the hash of q for each table takes times $O(\ell \cdot k)$.

Once the hash of q is computed, we need to check it against other values in the same bucket. This is where the computational cost of the distance function is important. In expectation, the bucket has at most a constant number of inhabitant . Thus, computing the distance of q with the other elements in the bucket takes an additional time of order $O(\ell \cdot d)$. The overall complexity is thus $O(\ell \cdot (k + d))$. Removing lower order terms we get that the final complexity is:

$$O(d \cdot n^{1/3})$$

1b: We need to hash q for ℓ tables. Each hash computation require k smaller hash computation, thus we need $O(\ell \cdot k)$ hash call. This is:

$$O\left(n^{1/3} \ln(n)^2\right)$$

2 Suppose you have a database with a set $P \subseteq \mathbb{R}^d$ of n items that are equipped with a distance function $\text{dist} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ satisfying the following sparsity condition:

$$|\{p \in P : \text{dist}(p, q) \leq 2\}| \leq 10\,.$$

Further assume that you have a $(r, c \cdot r, p_1, p_2)$-LSH hash family $\mathcal{H}$ for the considered distance function with parameters $r = 1$, $c = 2$, $p_1 = 1/2$ and $p_2 = 1/8$. That is,

$$\text{dist}(p, q) \leqslant 1 \implies \Pr[h(p) = h(q)] \geqslant 1/2$$
$$\text{dist}(p, q) \geqslant 2 \implies \Pr[h(p) = h(q)] \leqslant 1/8$$

where the probabilities are over $h \sim \mathcal{H}$.

Exploit the sparsity condition to modify the ANNS(c, r) construction seen in class so as to obtain a structure with the *same* asymptotic preprocessing and query times, but with the following improved guarantee:

> On query $q \in \mathbb{R}^d$, if $\min_{p \in P} \text{dist}(p, q) \leq 1$, then we return $\arg\min_{p \in P} \text{dist}(p, q)$ with probability close to 1.

(Notice that this is stronger than the guarantee seen in class as in that case one is only guaranteed to return a point p' such that $\text{dist}(p', q) \leq c \cdot r$ with probability close to 1.)

What is the preprocessing time, query time, and space requirement of your solution?

Solution: The solution that we use in here is very similar to the algorithm used in the lecture notes. In fact the preprocessing step is exactly the same. The only difference is that given a query q, we first compute $f_i(q)$ for all $1 \leq i \leq \ell$ and among all the points p that have the same hash function (i.e., $f_i(q) = f_i(p)$) we return the one with smallest distance to q. Now let p be the closest point to q. As shown in the lecture note, the probability that $f_i(q) = f_i(p)$ for some $1 \leq i \leq \ell$ is $1 - 1/n$. Therefore with a high probability we find the desired point. Now we need to discuss the preprocessing time, query time and the space complexity of our approach. It is easy to see that the space complexity and the preprocessing time is exactly same as the one described in the lecture note. For any query point q we need to spend $O(\ell \cdot k)$ time to compute $f_i(q)$ for all $1 \leq i \leq \ell$ (we assume the computation of the hash functions take constant time). For any candidate close point p we spend $O(d)$ time to calculate $dist(p, q)$. Notice that as shown in the lecture note, the probability of checking a point p with $dist(p, q) > 2$ is at most $1/n$. Therefore, the expected number of such points that we consider for each $1 \leq i \leq \ell$ is one. Therefore, given the sparsity assumption, for each such i, we check at most 11 points in our data set. Therefore the total number of the time that we compute $dist(p, q)$ is $O(\ell)$ which shows that the running time for each query is $O(\ell(k + d))$. Recall that $\ell = O(n^{\frac{\log 1/p_1}{\log 1/p_2}} \log n) = O(n^{1/3} \log n)$ and $k = O(\log n / \log(1/p_2)) = O(\log n)$.

3 Consider a submodular function $f : 2^N \to \mathbb{R}$ over the ground set $N = \{1, 2, 3, 4\}$. What is the value of the Lovàsz extension $\hat{f}(0.75, 0.3, 0.2, 0.3)$ as a function of f?

Solution: By the definition of $\hat{f}$ (see Lecture Notes 19) we get

$$\hat{f}(0.75, 0.3, 0.2, 0.3, 0) = 0.25f(\emptyset) + 0.45f(\{1\}) + 0.1f(\{1, 2, 4\}) + 0.2f(\{1, 2, 3, 4\}).$$

4 Hypergraph cuts. Let $G = (V, E)$ be a hypergraph with vertex set V and hyperedge set E (every hyperedge $e \in E$ is a subset of V; see Fig. 1 for an illustration). For $S \subseteq V$ the set of hyperedges crossing the cut $(S, V \setminus S)$ is defined as

$$E(S, V \setminus S) = \{e \in E : e \cap S \neq \emptyset \text{ and } e \cap V \setminus S \neq \emptyset\},$$

and the size of the cut $(S, V \setminus S)$ as $|E(S, V \setminus S)|$.

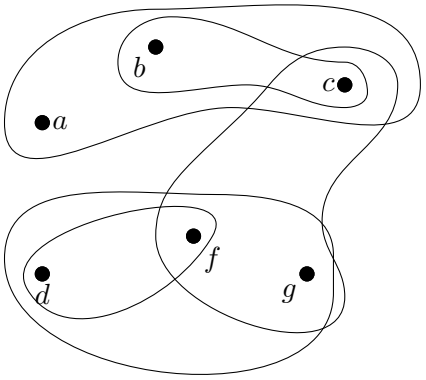

Figure 1. A hypergraph $G = (V, E)$ with $V = \{a, b, c, d, f, g\}$ and hyperedge set $E = \{e_1, e_2, e_3, e_4, e_5\}$, where $e_1 = \{a, b, c\}, e_2 = \{b, c\}, e_3 = \{d, f\}, e_4 = \{c, f, g\}$ and $e_5 = \{d, f, g\}$.

4a Give an algorithm that finds the size of the minimum cut in a given hypergraph G, i.e. outputs

$$\min_{S \subset V, S \neq \emptyset} |E(S, V \setminus S)|.$$

For example, the size of the minimum cut in the hypergraph G in Fig. 1 is 1. There are two minimum cuts: $(\{a\}, \{b, c, d, f, g\})$ and $(\{a, b, c\}, \{d, f, g\})$.

Your algorithm should run in time polynomial in the number of vertices and hyperedges in G.

Solution: Let $f(S) = E(S, V \setminus S)$. We first observe that f is submodular. We verify the diminishing returns property. For every $S \subseteq T \subseteq V$ and $u \in V$ we have

$$\begin{aligned} f(u|S) &= f(S \cup \{u\}) - f(S) \\ &= |\{e \in E : u \in e, e \cap S = \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) \neq \emptyset\}| \\ &- |\{e \in E : u \in e, e \cap S \neq \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) = \emptyset\}| \end{aligned} \tag{1}$$

Since $S \subseteq T$, we have

$$\{e \in E : u \in e, e \cap T = \emptyset \text{ and } e \cap V \setminus (T \cup \{u\}) \neq \emptyset\} \subseteq \{e \in E : u \in e, e \cap S = \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) \neq \emptyset\}$$

and

$$\{e \in E : u \in e, e \cap S \neq \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) = \emptyset\} \subseteq \{e \in E : u \in e, e \cap T \neq \emptyset \text{ and } e \cap V \setminus (T \cup \{u\}) = \emptyset\}.$$

We thus get by (1)

$$\begin{aligned} f(u|S) &= |\{e \in E : u \in e, e \cap S = \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) \neq \emptyset\}| \\ &\quad - |\{e \in E : u \in e, e \cap S \neq \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) = \emptyset\}| \\ &\leq |\{e \in E : u \in e, e \cap T = \emptyset \text{ and } e \cap V \setminus (T \cup \{u\}) \neq \emptyset\}| \\ &\quad - |\{e \in E : u \in e, e \cap S \neq \emptyset \text{ and } e \cap V \setminus (S \cup \{u\}) = \emptyset\}| \\ &\geq f(u|T) \end{aligned}$$

and therefore $f(S)$ is submodular.

Now we would like to use the fact that unconstrained submodular function minimization can be done in polynomial time. However, we do have constraints $S \neq \emptyset$ and $S \neq V$. Thus, instead we pick an element $u \in V$ and then for every other $v \in V \setminus \{u\}$ define

$$g(S) = f(S \cup \{u\})$$

and find

$$\min_{S \subseteq V \setminus \{u,v\}} g(S).$$

Note that $g(S)$ is submodular, since for every $w \in V \setminus \{u, v\}$ and $S \subseteq T \subseteq V$ one has

$$g(w|S) = f(w|S \cup \{u\}) \geq f(w|T \cup \{u\}) = g(w|T)$$

by the diminishing returns property of f.

4b Give a randomized polynomial time algorithm that outputs, given a hypergraph G where every hyperedge contains three vertices, a cut $(S, V \setminus S)$ such that

$$\mathbb{E}[|E(S, V \setminus S)|] \geq (3/4)OPT, \qquad (*)$$

where

$$OPT = \max_{S \subseteq V} |E(S, V \setminus S)|.$$

Note that unlike **4a**, here we are interested in the **maximum** cut. Your algorithm should run in time polynomial in the number of vertices and hyperedges in G, and you should prove that the expected size of the cut that it outputs satisfies $(*)$.

Solution: Let S include every vertex $u \in V$ independently with probability 1/2. Then

$$\mathbb{E}[|E(S, V \setminus S)|] = \sum_{e \in E} \Pr[e \in E(S, V \setminus S)].$$

Let $e = \{u, v, w\}$, and suppose that $u \in S$ (this is without loss of generality, as $E(S, V \setminus S) = E(V \setminus S, S)$). Then

$$\Pr[e \in E(S, V \setminus S)] = 1 - \Pr[v, w \in S] = 3/4.$$

Thus, a random cut cuts at least 3/4 of the hyperedges in expectation. By Markov's inequality applied to $|E| - |E(S, V \setminus S)|$ we have

$$\Pr[|E| - |E(S, V \setminus S)| > 1/4(1 + 1/n)|E|] \leq 1/(1 + 1/n) = 1 - O(1/n).$$