Exercise Set XI, Algorithms II 2024

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.

Additional problems on LSH and Nearest Neighbor

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

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

and $\mathcal{H}_2$ satisfies

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

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,

$$\begin{aligned} \text{dist}(p, q) \leqslant 1 &\implies \mathbb{P}\left[h(p) = h(q)\right] \geqslant 1/2 \\ \text{dist}(p, q) \geqslant 2 &\implies \mathbb{P}\left[h(p) = h(q)\right] \leqslant 1/8 \end{aligned}$$

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 $k = O(\log n / \log(1/p_2)) = O(\log n)$.

Submodularity

3 Consider two submodular functions seen in class: the cut function $\delta(\cdot)$ of a graph and the coverage function $c(\cdot)$ of finite collection of sets. Are they monotone? Give proofs or counterexamples. *(We say that a set function f is* monotone *if $f(B) \geq f(A)$ for all $A \subseteq B \subseteq N$.)*

Solution:

- The cut function $\delta(\cdot)$ of a graph is not monotone. To see this, consider a graph with two vertices $\{a, b\}$ connected by an edge. Then $0 = \delta(\{a, b\}) < \delta(\{a\}) = 1$.
- The coverage function is monotone. Let $T_1, \ldots, T_n$ be a finite collection of finite sets, and let c be the coverage function defined on subsets of $[n]$ so that $c(S) = |\cup_{i \in S} T_i|$. Let $A \subseteq B \subseteq [n]$ be two arbitrary subsets. Clearly, $\cup_{i \in A} T_i \subseteq \cup_{i \in B} T_i$, and hence $f(A) = |\cup_{i \in A} T_i| \leq |\cup_{i \in B} T_i| = f(B)$.

4 Let $f_1, \ldots, f_k : 2^N \to \mathbb{R}$ be submodular functions on the ground set N. Show that if $\lambda_1, \lambda_2, \ldots, \lambda_k \geq 0$ then the function $g : 2^N \to \mathbb{R}$ defined by $g(S) = \sum_{i=1}^{k} \lambda_i f_i(S)$ is a submodular function.

Solution: Let A, B be any two subsets of N. Since $f_1, \ldots, f_k$ are submodular functions, for all $i = 1, \ldots, k$, we have that:

$$f_i(A) + f_i(B) \geq f_i(A \cup B) + f_i(A \cap B).$$

Taking the weighted sum over all $i = 1, \ldots, k$, we get,

$$\underbrace{\sum_{i=1}^{k} \lambda_i f_i(A)}_{g(A)} + \underbrace{\sum_{i=1}^{k} \lambda_i f_i(B)}_{g(B)} \geq \underbrace{\sum_{i=1}^{k} \lambda_i f_i(A \cup B)}_{g(A \cup B)} + \underbrace{\sum_{i=1}^{k} \lambda_i f_i(A \cap B)}_{g(A \cap B)}.$$

Thus $g(A) + g(B) \geq g(A \cup B) + g(A \cap B)$ for all $A, B \subseteq N$, and hence g is a submodular function.

5 In the MaxSAT problem, we are given a set of m disjunctions over n variables and their negations. That is, each clause is of the form:

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_d\,,$$

where each ℓ_i is either a variable or the negation of a variable. The goal is to assign each variable a value true or false, so that as many clauses as possible are true. Show how to formulate MaxSAT as a submodular optimization problem with a single matroid constraint.

Solution: Let the ground set N be the set of all n variables, together with their negations. We impose a partition matroid constraint on the ground set, that for each variable x allows at most one of each $\{x, \neg x\}$ to be selected (this corresponds to setting x to be true or false, respectively). Now, for any selected set of literals S, let $f(S)$ the number of clauses containing at least one literal from S. This is precisely the number of clauses that are satisfied by the truth assignment that sets x to true if $x \in S$ is selected and false if $\neg x \in S$. But f is now simply a coverage function: let T_x and $T_{\neg x}$ be the set of all clauses containing x and $\neg x$. Then $f(S) = \left|\bigcup_{\ell \in S} T_\ell\right|$. Thus, f is submodular.

6 *(Homework problem previous year)* You have just finished developing your first really cool App. Your target group are the students at EPFL and you plan to sell the App *expensively*. However, to launch your application (and to spread the word about how cool it is), you are willing to give away k free copies. To maximize the influence of these free copies you have the following graph model:

- The "friendship" graph has a vertex v for each student at EPFL and an edge $\{u, v\}$ if u and v are friends.
- For each student v, there is a probability $0 \leq p_v \leq 1$ that models how likely v is to tell their friends about your App (assuming v knows about it).

Given this information, the goal is to give free copies to k students so as to maximize the expected number of students that will have heard about your App. More formally, let A be a random set of students that contains each student v with probability p_v independently. Then the goal is to find a set $S = \{u_1, \ldots, u_k\}$ of k students so as to maximize

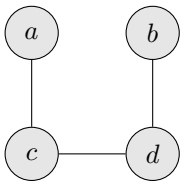
$$f(S) = \mathbb{E}_A \left[|\{v : v \text{ is informed if students in } A \text{ are spreading the word}\}| \right] ,$$

where v is informed if there is a path from v to a vertex in S in which all vertices, except possibly v, are in A. For an example see below.

Prove that the function f is submodular. Thus your problem reduces to that of maximizing a submodular function subject to a cardinality constraint. Since f is clearly monotone, we can use the greedy algorithm explained in class to launch our product in an awesome way.

Hint: show first that f is submodular if $p_v \in \{0, 1\}$ for every student v. Then notice that any probability assignment $p : V \to [0, 1]$ can be written as a weighted sum of such probability assignments.

Example. We have students a, b, c, d with $p_a = 1/2$, $p_b = 1/3$, $p_c = 0$, $p_d = 1$ and the friendship graph is



We are going to evaluate $f(\{a, b\})$. Notice that since c is never in the set A since $p_c = 0$ and d is always in A since $p_d = 1$ we have four potential outcomes of A. We consider each outcome separately:

Case 1 $A = \{d\}$**:** This happens with probability $(1 - p_a)(1 - p_b) = 1/3$ and students $\{a, b\}$ will be informed about your App.

Case 2 $A = \{d, a\}$**:** This happens with probability $p_a(1 - p_b) = 1/3$ and students $\{a, b, c\}$ will be informed about your App.

Case 3 $A = \{d, b\}$**:** This happens with probability $(1 - p_a)p_b = 1/6$ and students $\{a, b, c, d\}$ will be informed about your App.

Case 4 $A = \{d, a, b\}$**:** This happens with probability $p_a p_b = 1/6$ and students $\{a, b, c, d\}$ will be informed about your App.

Hence, in this example we have

$$f(\{a, b\}) = \underbrace{\frac{1}{3} \cdot 2}_{\text{Case 1}} + \underbrace{\frac{1}{3} \cdot 3}_{\text{Case 2}} + \underbrace{\frac{1}{6} \cdot 4}_{\text{Case 3}} + \underbrace{\frac{1}{6} \cdot 4}_{\text{Case 4}} = 3 .$$

Solution:

We first show that f is submodular if $p_v \in \{0, 1\}$ for all $v \in V$. Let $\mathcal{A}$ be the set of vertices such that $p_v = 1$, and let $f_{\mathcal{A}}(S)$ be the number of vertices v that are reachable from S through a path p entirely contained in $\mathcal{A}$ besides possibly v. To prove that f is submodular for a deterministic set $\mathcal{A}$, we show that for $T \subseteq S \subseteq V$ and $u \in V \backslash S$, $f_{\mathcal{A}}(u|S) \leq f_{\mathcal{A}}(u|T)$.[1]

Recall that $f_{\mathcal{A}}(u|S) = f_{\mathcal{A}}(u \cup S) - f_{\mathcal{A}}(S)$ which basically counts the number of vertices v reachable from u and not from any other vertex in S through a path contained entirely in $\mathcal{A}$ (besides possibly v). Similarly, $f_{\mathcal{A}}(u|T) = f_{\mathcal{A}}(u \cup T) - f_{\mathcal{A}}(T)$ counts the number of vertices v reachable from u and not from any other vertex in T through a path contained entirely in $\mathcal{A}$(besides possibly v).

However, $T \subseteq S$, so the vertices reachable from T through a path entirely contained in $\mathcal{A}$ are also reachable from S through a path contained entirely in $\mathcal{A}$. Thus, the number of new vertices that become reachable through a path entirely contained in $\mathcal{A}$ due to the addition of u to S is at most that resulting from the addition of u to T. So, $f_{\mathcal{A}}(u|S) \leq f_{\mathcal{A}}(u|T)$, and $f_{\mathcal{A}}$ is submodular.

Now, let $p_v : V \to [0, 1]$, and let $w(\mathcal{A})$ be the probability of obtaining the set A. Then,

$$f(S) = \sum_{\mathcal{A} \subseteq V} w(\mathcal{A}) f_{\mathcal{A}}(S) \tag{1}$$

and since $w(\mathcal{A})) \geq 0$, we have that f is submodular by Problem 5.

[1] We remark that an alternative proof is to observe that $f_{\mathcal{A}}$ is simply a coverage function.