ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Final Exam, Advanced Algorithms 2016-2017

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudo-code without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- **You are allowed to refer to material covered in the course** including theorems without reproving them.
- **Do not touch until the start of the exam.**

Good luck!

Name: _____ **N° Sciper:** _____

Problem 1 / 20 points	Problem 2 / 20 points	Problem 3 / 20 points	Problem 4 / 20 points	Problem 5 / 20 points

Total / 100

1 *(consisting of subproblems **a**-**b**, 20 pts)* **Basic questions.** This problem consists of two subproblems that are each worth 10 points.

1a *(10 pts)* **LSH for Jaccard similarity.**

Recall the Jaccard index that we saw in Exercise Set 10: Suppose we have a universe U. For non-empty sets $A, B \subseteq U$, the Jaccard index is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}\,.$$

Design a locality sensitive hash (LSH) family $\mathcal{H}$ of functions $h : 2^U \to [0,1]$ such that for any non-empty sets $A, B \subseteq U$,

$$\Pr_{h\sim\mathcal{H}}[h(A) \neq h(B)] \begin{cases} \leq 0.01 & \text{if } J(A,B) \geq 0.99, \\ \geq 0.1 & \text{if } J(A,B) \leq 0.9. \end{cases}$$

(In this problem you are asked to explain the hash family and argue that it satisfies the above properties. Recall that you are allowed to refer to material covered in the course.)

Solution: Let us describe $\mathcal{H}$ by giving a procedure to sample an element $h \in \mathcal{H}$:

- for each $u \in U$, sample h_u uniformly at random from $[0,1]$.
- set $h(A) = \min_{u\in A} h_u$ for any non-empty $A \subseteq U$ (i.e., MinHashing).

In Exercise Set 10, we showed that $\Pr[h(A) = h(B)] = J(A,B)$. So $\Pr[h(A) \neq h(B)] = 1 - J(A,B)$ and the claimed bounds follow immediately.

1b *(10 pts)* **Randomized algorithms for min-cut.** In class, we saw Karger's beautiful randomized algorithm for finding a min-cut in an undirected graph $G = (V, E)$ with $n = |V|$ vertices. Each iteration of Karger's algorithm can be implemented in time $O(n^2)$, and if repeated $\Theta(n^2 \log n)$ times, Karger's algorithm returns a min-cut with probability at least $1 - 1/n$. However, this leads to the often prohibitively large running time of $O(n^4 \log n)$.

Karger and Stein made a crucial observation that allowed them to obtain a much faster algorithm for min-cut: the Karger-Stein algorithm runs in time $O(n^2 \log^3 n)$ and finds a min-cut with probability at least $1 - 1/n$.

Explain in a couple of sentences the main idea that allowed Karger and Stein to modify Karger's algorithm into the much faster Karger-Stein algorithm. In other words, what are the main differences between the two algorithms?

Solution: The probability of contracting an edge in a fixed min-cut C is much lower in the initial contractions (in Karger's algorithm) than at the end. Thus it makes sense to repeat the later contractions more times than the initial ones. This is the main idea of the Karger-Stein algorithm.

Their implementation of this idea is to perform contractions so as to reduce the size of the graph from n to $n/\sqrt{2}$ (they make $n - n/\sqrt{2}$ contractions). Then find a min-cut recursively in the smaller graph by repeating the same algorithm twice on the smaller graph and choose the smaller of the obtained cuts. (These recursive calls will also split and so on.) The running time of this algorithm is roughly the same as one iteration of Karger's algorithm but the success probability is much higher.

2 *(20 pts)* **Set packing in the semi-streaming model.**

Consider the problem of finding a maximum cardinality set packing in the semi-streaming model. An instance of this problem consists of a known universe U of n elements and sets $S \subseteq U$ are streamed one-by-one. The goal is to select a family $\mathcal{T}$ of pairwise disjoint sets (i.e., $S \cap S' = \emptyset$ for any two distinct sets $S, S' \in \mathcal{T}$) of maximum cardinality while only using $O(n \cdot \operatorname{poly} \log n)$ storage space.

Devise an algorithm in this setting that returns a set packing of cardinality at least $1/k$ times that of a maximum cardinality set packing, assuming that each streamed set S has cardinality at most k, i.e., $|S| \leq k$.

(In this problem you are asked to (i) design the algorithm, (ii) show that it uses $O(n \cdot \operatorname{polylog} n)$ space, and (iii) prove that it returns a solution of cardinality at least $1/k$ times the cardinality of a maximum cardinality set packing. Recall that you are allowed to refer to material covered in the course.)

Solution:

We run the simple greedy algorithm:

1. Initially, let $\mathcal{T} = \emptyset$.
2. For each streamed S: if S is disjoint from all sets in $\mathcal{T}$, add S to $\mathcal{T}$.
3. At the end, we simply return $\mathcal{T}$ as our solution.

We now analyze the greedy algorithm in terms of space and approximation guarantee.

Space: Since at all times $\mathcal{T}$ is a family of disjoint sets, we have $\sum_{S \in \mathcal{T}} |S| \leq n$. If we store each selected set S as a list of its elements this will take space $|S| \log n$ for each set $S \in \mathcal{T}$ (the $\log n$ is the space required to save the identifier of each element). Thus, as $\sum_{S \in \mathcal{T}} |S| \leq n$, we require $O(n \log n)$ space in total.

Approximation ratio: Let $\mathcal{O}$ be an optimal set packing. The greedy algorithm returns a maximal set packing so any $O \in \mathcal{O}$ intersects at least one set in $\mathcal{T}$ (maybe itself if it was selected by greedy). Moreover, any set S can intersect at most k sets in $\mathcal{O}$ since $|S| \leq k$. Therefore,

$$|\mathcal{O}| = \sum_{O \in \mathcal{O}} 1 \leq \sum_{O \in \mathcal{O}} \sum_{S \in \mathcal{T} : S \cap O \neq \emptyset} 1 = \sum_{S \in \mathcal{T}} \sum_{O \in \mathcal{O} : S \cap O \neq \emptyset} 1 \leq \sum_{S \in \mathcal{T}} k = k|\mathcal{T}| \,.$$

(This is very similar to Problem 4 in Exercise Set 10.)

3 *(20 pts)* **Amplifying success probability of an unbiased estimator.**

Professor Ueli von Gruyères has worked intensely throughout his career to get a good estimator of the yearly consumption of cheese in Switzerland. Recently, he had a true breakthrough. He was able to design an incredibly efficient randomized algorithm $\mathcal{A}$ that outputs a random value X satisfying

$$\mathbb{E}[X] = c \qquad \text{and} \qquad \text{Var}[X] = c^2\,,$$

where c is the (unknown) yearly consumption of cheese in Switzerland. In other words, $\mathcal{A}$ is an unbiased estimator of c with variance c^2.

Use Ueli von Gruyères' algorithm $\mathcal{A}$ to design an algorithm that outputs a random value Y with the following guarantee:

$$\Pr[|Y - c| \geq \epsilon c] \leq \delta \qquad \text{where } \epsilon > 0 \text{ and } \delta > 0 \text{ are small constants.} \tag{1}$$

Your algorithm should increase the resource requirements (its running time and space usage) by at most a factor $O(1/\epsilon^2 \cdot \log(1/\delta))$ compared to the requirements of $\mathcal{A}$.

(In this problem you are asked to (i) design the algorithm using $\mathcal{A}$, (ii) show that it satisfies the guarantee (1)*, and (iii) analyze how much the resource requirements increase compared to that of simply running $\mathcal{A}$. Recall that you are allowed to refer to material covered in the course.)*

Solution: The idea of the algorithm is to first decrease the variance by taking the average of $t = 10/\epsilon^2$ independent runs of $\mathcal{A}$. We then do the median trick. Formally, consider the algorithm $\mathcal{B}$ that runs t independent copies of $\mathcal{A}$ and then outputs the average of the t estimates obtained from the independent runs of $\mathcal{A}$. Let B be the random output of this algorithm. As seen in class, we have $\mathbb{E}[B] = c$ (it is still an unbiased estimator) and $\text{Var}[B] = c^2/t$. Now by Chebychev's Inequality we have

$$\Pr[|B - c| \geq \epsilon c] \leq \frac{\text{Var}[B]}{\epsilon^2 c^2} = \frac{1}{t\epsilon^2} = 1/10 \qquad \text{(since we selected } t = 10/\epsilon^2)\,.$$

So algorithm $\mathcal{B}$ returns a $1 \pm \epsilon$ approximation with probability at least $9/10$. We now want to decrease the probability $1/10$ of failing all the way down to δ. To do this we use the median trick. Let $\mathcal{C}$ be the algorithm that runs $u = 10\ln(1/\delta)$ independent copies of $\mathcal{B}$ and outputs the *median* of the obtained copies. Let Y be the random output of $\mathcal{C}$. We now analyze the failure probability of $\mathcal{C}$, i.e., we wish to show $\Pr[|Y - c| \geq \epsilon c] \leq \delta$. To do so define $Z_i \in \{0, 1\}$ to be the indicator random variable that takes value 1 if the i:th run of $\mathcal{B}$ outputs a value B_i such that $|B_i - c| \geq \epsilon c$. Note that $\Pr[|B_i - c| \geq \epsilon c] \leq 1/10$ and so $\Pr[Z_i = 1] \leq 1/10$. So if we let $Z = Z_1 + Z_2 + \cdots + Z_u$, then Z is a sum of independent variables where $\mathbb{E}[Z] \leq u/10$. Moreover since Y is the median of the independent runs of $\mathcal{B}$,

$$\Pr[|Y - c| \geq \epsilon c] \leq \Pr[Z \geq u/2]\,.$$

We shall now analyze $\Pr[Z \geq u/2]$ using the Chernoff Bounds. Indeed, since Z is a sum of *independent* random variables taking values in $\{0, 1\}$ we have

$$\Pr[Z \geq u/2] \leq \Pr[Z > 3 \cdot \mathbb{E}[Z]] \leq e^{-\ln(1/\delta)} = \delta\,.$$

We have thus proved that $\mathcal{C}$ satisfies the right guarantees. Let us know analyze its resource requirements. $\mathcal{C}$ runs $O(\log(1/\delta)$ copies of $\mathcal{B}$ and each copy of $\mathcal{B}$ runs $O(1/\epsilon^2)$ copies $\mathcal{A}$. Thus the total resource requirements increase by at most a factor $O(\log(1/\delta)1/\epsilon^2)$ as required (calculating the mean and median can be done in linear time so it does not affect the asymptotic running time).

4 *(20 pts)* **Min-weight perfect matching via determinants.** Consider the following algorithm that takes as input a complete n-by-n bipartite graph $G = (U \cup V, E)$ with positive integer edge-weights $w : E \to \mathbb{Z}_{>0}$:

> MinWeightPerfectMatching(G, w):
>
> 1. **for** each edge $e \in E$ (i.e., each pair (u, v) since the graph is complete)
> 2. select independently and uniformly at random $p(e) \in \{1, \ldots, n^2\}$.
> 3. Define a bi-adjacency matrix A with n rows (one for each $u \in U$) and n columns (one for each $v \in V$) as follows:
>
> $$A_{u,v} = 2^{n^{100} w(u,v)} \cdot p(u, v)\,.$$
>
> 4. **return** largest positive integer i such that $2^{i \cdot n^{100}}$ divides $\det(A)$ (if no such i exists, we return 0).

Prove that the above algorithm returns the value of a min-weight perfect matching with probability at least $1 - 1/n$. Recall that you are allowed to refer to material covered in the course.

Hint: Let $\mathcal{M}_i$ denote the set of perfect matchings M whose weight $\sum_{e \in M} w(e)$ equals i. Use that one can write $\det(A)$ as follows:

$$\det(A) = \sum_{i=0}^{\infty} 2^{i \cdot n^{100}} f_i(p) \qquad \text{where } f_i(p) = \sum_{M \in \mathcal{M}_i} \text{sign}(M) \prod_{e \in M} p(e)\,.$$

Here $\text{sign}(M) \in \{\pm 1\}$ is the sign of the permutation corresponding to M.

Solution: Let i^* be the weight of a minimum-weight perfect matching (and so $\mathcal{M}_{i^*}$ contains all perfect matchings of minimum weight). We first prove that

$$\Pr[f_{i^*}(p) = 0] \leq 1/n\,, \tag{2}$$

where the probability is over the random selection of $p(e)$'s in Step 2 of the algorithm. First note that if we think of $f_{i^*}(x)$ as a polynomial in the variables $x(e)$ for each edge $e \in E$ (that are replaced by the values $p(e)$), then we have that $f_{i^*}(x)$ is not identical to zero since $\mathcal{M}_{i^*} \neq \emptyset$ and the monomial $\prod_{e \in M} x(e)$ corresponding to a matching $M \in \mathcal{M}_{i^*}$ appears only once (with a non-zero coefficient) in $f_{i^*}(p)$. The probability bound (2) now follows from the Schwartz-Zippel lemma that we saw in class since each $p(e)$ is selected from a set of n^2 values at random and the degree of f_{i^*} is n.

We now prove that the algorithm *always* outputs the correct answer if $f_{i^*}(p) \neq 0$ (and hence with probability at least $1 - 1/n$ over the selection of $p(e)$'s). First note that $2^{i^* n^{100}}$ divides $\det(A)$, since for $i < i^*$ we have $f_i(x) = 0$ (in particular, $f_i(p) = 0$) because i^* is the value of a min-weight perfect matching and so $\mathcal{M}_i = \emptyset$, and all $f_i(p)$ (for all i) are integers, therefore

$$\det(A) = \sum_{i=0}^{\infty} 2^{i \cdot n^{100}} f_i(p) = \sum_{i=i^*}^{\infty} 2^{i \cdot n^{100}} f_i(p)$$

is divisible by $2^{i^* \cdot n^{100}}$. We have thus proved that the algorithm outputs an integer that is at least i^*. We now show that if $f_{i^*}(p) \neq 0$, then $2^{(i^*+1) n^{100}}$ does not divide $\det(A)$ and thus the algorithm must output the correct value.

To do so, we bound the absolute value of $f_{i^*}(p)$. We have

$$|f_{i^*}(p)| = \left| \sum_{M \in \mathcal{M}_i} \text{sign}(M) \prod_{e \in M} p(e) \right| \leq \sum_{M \in \mathcal{M}_{i^*}} \prod_{e \in M} p(e) \leq |\mathcal{M}_{i^*}| \prod_{e \in M} n^2 \leq n! \cdot (n^2)^n \leq n^{3n} < 2^{n^{100}}.$$

Therefore

$$\left| 2^{i^* \cdot n^{100}} f_{i^*}(p) \right| < 2^{i^* \cdot n^{100}} 2^{n^{100}} = 2^{(i^*+1) \cdot n^{100}}$$

but $2^{i^* \cdot n^{100}} f_{i^*}(p) \neq 0$, and so $2^{(i^*+1)n^{100}}$ does not divide $2^{i^* \cdot n^{100}} f_{i^*}(p)$, whereas it does divide $2^{i \cdot n^{100}} f_i(p)$ for all $i > i^*$. Therefore it does not divide $\det(A)$.

5 *(20 pts)* **Assigning vertex potentials.** Design and analyze a polynomial time algorithm for the following problem:

INPUT: An undirected graph $G = (V, E)$.

OUTPUT: A non-negative vertex potential $p(v) \geq 0$ for each vertex $v \in V$ such that

$$\sum_{v \in S} p(v) \leq |E(S, \bar{S})| \quad \text{for every } \emptyset \neq S \subsetneq V \quad \text{and} \quad \textstyle\sum_{v \in V} p(v) \text{ is maximized.}$$

(Recall that $E(S, \bar{S})$ denotes the set of edges that cross the cut defined by S, i.e., $E(S, \bar{S}) = \{e \in E : |e \cap S| = |e \cap \bar{S}| = 1\}$.)

Hint: formulate the problem as a large linear program (LP) and then show that the LP can be solved in polynomial time.

(In this problem you are asked to (i) design the algorithm, (ii) show that it returns a correct solution and that it runs in polynomial time. Recall that you are allowed to refer to material covered in the course.)

This year we didn't cover the Elliposid method which is necessary for solving the above problem. By using the Ellipsoid method, the above reduces to the following: Given p, give an efficient algorithm that either verifies that p is feasible or outputs a violated constraint.

Solution: Consider the following linear program with a variable $p(v)$ for each vertex $v \in V$:

$$\begin{aligned} \max \quad & \sum_{v \in V} p(v) \\ \text{subject to} \quad & \sum_{v \in S} p(v) \leq |E(S, \bar{S})| \qquad \text{for all } \emptyset \subset S \subset V \\ & p(v) \geq 0 \qquad \text{for all } v \in V \end{aligned}$$

We show that this linear program can be solved in polynomial time using the Ellipsoid method by designing a polynomial time separation oracle. That is we need to design a polynomial time algorithm that given $p^* \in \mathbb{R}^V$ certifies that p^* is a feasible solution or outputs a violated constraints.

The non-negativity constraints are trivial to check in time $O(|V|)$ (i.e., polynomial time) so let us worry about the other constraints. These constraints can be rewritten as $f(S) \geq 0$ for all $\emptyset \subset S \subset V$, where

$$f(S) = |E(S, \bar{S})| - \sum_{v \in S} p^*(v) \quad \text{for } \emptyset \subseteq S \subseteq V.$$

Note that f is a submodular function since it is a sum of two submodular functions: the cut function (which is submodular as seen in class) and a linear function (trivially submodular). Hence there is a violated constraint if and only if

$$\min_{\emptyset \subseteq S \subset V} f(S) < 0\,.$$

This is an instance of the submodular function minimization problem with the exception that we do not allow $S = V$ for a solution. Therefore we solve n instances of submodular minimization on the smaller ground sets $V \setminus \{v_1\}$, $V \setminus \{v_2\}$, $\ldots$, $V \setminus \{v_n\}$ where $V = \{v_1, \ldots, v_n\}$. Since submodular function minimization is polynomial time solvable (and we can clearly evaluate our submodular function in polynomial time), we can solve the separation problem in polynomial time.