Princeton University
COS 521: Advanced Algorithms
Final Exam Fall 2021

Absolute last time to hand in (regardless of when you start): Dec 20 11:59pm.

Instructions: The test has 7 questions. Each question is worth 20 points. *Your final grade will sum your six highest scores* — you are explicitly encouraged to completely drop whatever question you find most challenging. Finish the test within **48 hours** after first reading it. No extensions will be granted without dean approval. You can consult any notes/handouts/readings from the class webpage (or last year's webpage) and feel free to quote, without proof (but with citation), any results from there. You **cannot** consult any other source or person in any way.

Do not read the test before you are ready to work on it.

Write and sign the honor code pledge on your exam (The pledge is "I pledge my honor that I have not violated the honor code during this examination."). You can use your initials to sign.

Matt, Mark, Linda, and Zhou will be available during the exam period to answer clarification questions *about the exam (we will not give hints or guidance). The best way to ask a question is to post a* private *Ed Question. We cannot guarantee exceptionally fast response times, so please ask your questions early during your 48-hour window.*

In case of unresolved doubt, try to explain your confusion as part of the answer and maybe you will receive partial credit. In general, stating clearly what you are trying to do can get you partial credit.

1. In Lecture, we designed a 3/4-approximation for MAX-2SAT using randomized rounding. In MAX-2SAT, each clause has 1 or 2 variables. Design an *extremely simple* (poly-time) randomized algorithm that obtains a 7/8-approximation for the special case of MAX-3SAT where every clause has *exactly* 3 (distinct) variables. Your algorithm description should be at most a few sentences. Your algorithm should take as input a MAX-3SAT instance (where every clause has exactly 3 distinct variables) and randomly output an assignment that satisfies $7\text{OPT}/8$ clauses in expectation (where OPT is the maximum number of clauses satisfied by any assignment), and terminate in time $\text{poly}(n, m)$ (n is the number of variables, m is the number of clauses).

2. In the Ellipsoid Algorithm Lecture, we wrote down the Held-Karp relaxation for the Traveling Salesman Problem. In the integer programming formulation, there are two linear constraints: one constraining that every node has exactly two incident edges, and the subtour elimination constraints. Come up with examples of graphs showing that both constraints are necessary. Specifically, design a weighted, undirected graph G_1 where the IP *without subtour elimination constraints* does not output the optimal tour. Design a weighted, undirected graph G_2 where the IP *that requires only that each node has* ≥ 2 *incident edges* (that is, $\sum_j X_{ij} = 2$ is replaced with $\sum_j X_{ij} \geq 2$, but the subtour elimination constraints are still there) does not output the optimal tour.

3. Prove von Neumann's min-max theorem: For all $n \times m$ matrices A, if x ranges over all distributions in the n-simplex (that is, $x_i \geq 0$ for all i, $\sum_i x_i = 1$), and y ranges over all distributions in the m-simplex, then: $\min_x \max_y x^T A y = \max_y \min_x x^T A y$. You may assume Strong LP Duality.

4. Prove that the following collections are matroids. Below, $\mathcal{U}$ denotes the universe of elements, and $\mathcal{I}$ denotes the collection of independent sets.

   - Let $\mathcal{I}'$ be any matroid over $\mathcal{U}$, and $k \geq 0$ be any constant. $\mathcal{I} = \{S \in \mathcal{I}', |S| \leq k\}$.
   - Let $\mathcal{I}'$ be any matroid over $\mathcal{U}$. A basis of $\mathcal{I}'$ is a set $T \in \mathcal{I}'$ such that for all $T' \supset T$,[1] $T' \notin \mathcal{I}'$. Define to $B(\mathcal{I}')$ to be the bases of $\mathcal{I}'$. $\mathcal{I} = \{S \subseteq \mathcal{U} | \exists T \in B(\mathcal{I}'),$ such that $S \cap T = \emptyset\}$.
   - $\mathcal{L}$ is any *laminar* family of subsets of $\mathcal{U}$. That is, for all $S, T \in \mathcal{L}$, at least one of the following holds: $S \subseteq T, T \subseteq S$, or $S \cap T = \emptyset$. For each $S \in \mathcal{L}$, there is a cap $c_S \geq 0$. $\mathcal{I} = \{T \subseteq \mathcal{U} |\ |T \cap S| \leq c_S\ \forall S \in \mathcal{L}\}$.

5. Suppose you have black-box access to a poly-time algorithm $\mathcal{A}$ that can approximate Independent-Set within a constant factor $C < 1$. That is, given as input an unweighted, undirected graph G on any number of nodes n, $\mathcal{A}(\mathcal{G})$ outputs an independent set S of nodes in G such that $|S| \geq C \cdot \alpha(G)$ (where $\alpha(G)$ is the size of the largest independent set in G), and does so in time $\text{poly}(n)$.[2] Design another algorithm, using $\mathcal{A}$, which also terminates in time $\text{poly}(n)$, but guarantees a constant-factor approximation of $C' > C$. (Hint: The instructor solution has $C' = \sqrt{C}$).

[1] Note that $T' \supset T$ means that T' *is a strict superset of* T.

[2] For a reminder: a set of nodes S is an independent set in G if there are no edges between any pair of nodes in S.

6. Consider the following online problem: you have a memory cache of size n. There are $m > n$ possible elements of disk that you need to access. When a request arrives, if it is already in cache, it costs zero. If it's not already in cache, you must move it to cache and pay cost 1 (and decide which element to kick out of cache). Note that the only decisions you must make is which element to kick out of cache - you must move the requested element to cache. In both problems below, assume that the cache is initially set to $\{1, \ldots, n\}$ for any algorithm (online or the offline optimum).

   (a) Prove that no deterministic online algorithm achieves a competitive ratio better than n. Also, prove that the following algorithms have arbitrarily bad competiive ratios (i.e. the competitive ratio is unbounded).

   - Last-In-First-Out: replace the page most recently moved to memory cache.
   - Frequency Count: maintain a count x_i for each page, the number of times it's been accessed. Replace the page with the lowest frequency count.

   (b) Prove that the following algorithms have competitive ratio exactly n:

   - Least Recently Used: replace the page that was least recently accessed.
   - First-In-First-Out: replace the page least recently moved to memory cache.

7. Recall the Job Scheduling problem on unrelated machines: there are n jobs and m machines. Machine j can process job i in time p_{ij}. The goal is to assign the jobs to machines to minimize the makespan (that is, the time until every machine finishes every job assigned to them). Specifically, find an allocation x of jobs to machines (where $x_{ij} = 1$ if job i is assigned to machine j) so as to minimize $\max_j \sum_i x_{ij} p_{ij}$. For this problem, the machines are strategic agents. Machine j's utility for the allocation x when paid Q is $Q - \sum_i x_{ij} p_{ij}$ (spending one unit of processing time costs them one dollar, and they want to maximize their payment minus cost).

   Design a truthful (dominant strategy truthful), deterministic mechanism that guarantees an m-approximation to the optimal makespan (that is, the makespan output is at most m times the makespan of the optimal schedule).