15-251: Great Theoretical Ideas In Computer Science

Fall 2013

Practice Test Solutions 2

Problem	Points	Score	Problem	Points	Score
1	7		5	20	
2	7		6	26	
3	7		7	26	
4	7				
			Total	100	

Part I: Short answers ($4 \times 7 = 28$ points)

Explanations provided in multiple choice questions are only for your understanding. You do not need to give any justification for these questions in the exam, unless specified otherwise.

1. [7 points]
In the version of the Gale-Shapley algorithm where men propose, what is the maximum number of men that can be matched with their last choice? Assume there are n men and n women. Hint: Recall the Gale-Shapley algorithm. Think about what happens if/when a man is matched with his last choice.

a) 0
✓b) 1
c) $\lfloor n/2 \rfloor$
d) $n - 1$

Why? It is easy to see that 1 man can get his last choice, at least when there is only one man and one woman. Further, when a man m is matched with the woman w of his last choice, all women other than w must be engaged with the men other than m. Hence, w must be unengaged when m proposed to her. Thus, no pairs are broken at this iteration, resulting in termination of the Gale-Shapley algorithm right afterwards.

2. [7 points]
If a subset of vertices S is matched by some matching (not necessarily maximal), is it true that we can always find some maximal (not necessarily maximum cardinality) matching that matches all vertices in S? Explain your answer in one sentence.

ANSWER: Yes. One can simply keep finding augmenting paths and swapping edges until the matching is maximal (in fact maximum). Note that swapping edges on an augmenting path does not unmatch any previously matched vertex.

3. [7 points]
In Homework 8, you designed a $\frac{7}{8}$ approximation algorithm for maximizing the number of satisfied clauses in 3-SAT. Explain in one sentence why this implies that any 3-SAT instance with at most 7 clauses is satisfiable. Recall that each variable appears at most once in each clause. This question does not require the knowledge of how the algorithm works.

ANSWER: Note that the algorithm from Homework 8 in fact satisfies at least $7/8$ fraction of all clauses. When the input formula has only 7 clauses, this would imply that the algorithm would satisfy at least $\lceil 7 \cdot 7/8 \rceil = 7$ clauses. Thus, the algorithm would always output a satisfying assignment, showing that every such formula must be satisfiable.

4. [7 points]
Consider the following problem: Given the graph of all cities of the US and edges with weights showing the time to go from one city to another, find a tour requiring minimum travel time that visits each city exactly once. Which of the following statement(s) are (known to be) true about this problem? (Assume $P \neq NP$. Multiple correct answers are possible.)
✓a) It is in P.
✓b) It is in NP.
c) It is NP-complete.
d) It is outside NP.

Why? Because this is a particular instance of TSP. It has a fixed solution. A program that outputs just this solution runs in polynomial time. Hence, the problem is in P. Further, a program that matches any given solution to this fixed solution is a polynomial-time verifier. Hence, the problem is also in NP. Since the problem is in P and we assume $P \neq NP$, option c) isn't checked.

Part II: (Variant of) Homework Question (20 points)

5. [20 points]
In Homework 8, you showed that for the maximum coverage problem, the greedy algorithm achieves $1 - 1/e$ approximation. Consider the following weighted generalization of the problem.

Weighted-Maximum-Cover (WMC): Consider a set of n elements $U = \{e_1, \ldots, e_n\}$. Let each element e_i have weight w_i, where $1 \leq i \leq n$. You are given t subsets $S_1, \ldots, S_t \subseteq U$. The task is to select k of the t subsets such that *the total weight of all distinct elements covered in their union is maximized*.

Observe that the original Maximum-Cover problem is a special case where the weight of every element is 1. In the weighted version, at every step the weighted greedy algorithm chooses a subset that has the maximum total weight of yet uncovered elements. Prove that even for WMC, the weighted greedy algorithm achieves a $1 - 1/e$ approximation. *You can prove simply the unweighted version from the homework to get half of the credit.*

ANSWER: Let OPT denote the total weight of the elements covered by the optimal solution, y_i denote the total weight of the elements covered by the greedy algorithm after selecting i subsets, and $z_i = OPT - y_i$. Note that initially, $y_0 = 0$ and $z_0 = OPT$.

Note that after the greedy algorithm selects i subsets, the weight of the elements covered by OPT but not yet covered by the greedy algorithm is at least z_i. Further, since k subsets of OPT cover the weight z_i, at least one of them must cover at least z_i/k of this weight. Since the greedy algorithm selects a subset in iteration $i+1$ holding the maximum uncovered weight, the subset picked would contribute at least z_i/k previously uncovered weight.

Hence,

$$\begin{aligned} & y_{i+1} \geq y_i + \frac{z_i}{k} \\ \Rightarrow \quad & OPT - y_{i+1} \leq OPT - y_i - \frac{z_i}{k} \\ \Rightarrow \quad & z_{i+1} \leq z_i \cdot \left(1 - \frac{1}{k}\right) \end{aligned}$$

Applying this repeatedly, we get

$$\begin{aligned} & z_i \leq \left(1 - \frac{1}{k}\right)^i \cdot OPT \\ \Rightarrow \quad & y_k = OPT - z_k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) OPT \geq \left(1 - \frac{1}{e}\right) OPT, \end{aligned}$$

as required.

Part III: Longer Problems ($2 \times 26 = 52$ points)

Remember that in this part, you need to formally prove your answer. Be crisp, precise, and accurate!

6. [26 points]
Consider the following BEST-SCHEDULE problem. You have k servers and n jobs. Job i requires a_i time to be processed (on any server), where $a_i \in \mathbb{N}$. You want to schedule each job on one of the servers. The processing time each server would take is equal to the sum of the processing times of the jobs it is assigned. Your goal is to find an assignment of the jobs to the servers to finish all the jobs as quickly as possible, i.e., to minimize the maximum of the processing times of all servers.

The decision version of the problem asks: Given $a_1, \ldots, a_n$, k, and an integer T, can jobs with processing times $a_1, \ldots, a_n$ can be scheduled on k servers such that all of them finish within time T. Prove that this is NP-complete. [Hint: To minimize the maximum processing time across all servers, you must try to equalize the processing times of all servers as much as possible, i.e., balance the load. Think about an NP-complete problem, one you already know, that fits this description.]

ANSWER: First, we see that the problem is in NP. Clearly, given the assignment of jobs to servers, one can verify whether all servers finish processing their assigned jobs within time T. Thus, the problem is in NP.

For NP-completeness, take an instance $e_1, \ldots, e_n$ of PARTITION. The question is to check if there exists $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} e_i = \sum_{i \in \{1,\ldots,n\} \setminus S} e_i$. Construct an instance of BEST-SCHEDULE as follows. Let there be 2 servers and n jobs. Set $a_i = e_i$ for $1 \leq i \leq n$. Further, set $T = (1/2) \cdot \sum_{i=1}^{n} e_i$. Clearly, the jobs can be scheduled on two servers so that all jobs finish within time T if and only if $e_1, \ldots, e_n$ can be partitioned into two subsets of equal sum $(1/2) \cdot \sum_{i=1}^{n} e_i$. Hence, the reduction is valid. Clearly, it is also a polynomial-time reduction.

Since PARTITION is NP-complete, we deduce that BEST-SCHEDULE is also NP-complete.

7. [26 points]
When the vertices of an undirected graph are partitioned into two sets, the set of edges with one endpoint in each partition (i.e., the edges between the partitions) are said to form a *cut*. Formally, given an undirected graph $G = (V, E)$, partition V into S and S' such that $S \cup S' = V$ and $S \cap S' = \emptyset$. Then, the set of all edges between S and S' form a *cut* of the graph. Now, the Max-Cut problem is defined as follows.

Max-Cut: Given an undirected graph G, find the maximum size of any cut of G.

Find a polynomial-time 2-approximation algorithm for this problem, and show that it achieves the required approximation. [Hint: Start from any partition of the vertex set. Then, gradually improve the number of edges between partitions by shifting one vertex at a time.]

ANSWER: The algorithm is as follows.

1. Start from any partition of the vertex set.
2. While there is a vertex with more edges across the partitions than within its own partition, move it to the other part.

First, note that when we move a vertex in the second step of the algorithm, the number of edges across partitions increases by at least 1. Hence, this algorithm must terminate after at most $|E|$ steps, making it a polynomial-time algorithm.

Further, at the end, at least half of the edges incident on every vertex are across partitions. Hence, at least $|E|/2$ of all edges must be across partitions, making this algorithm a 2-approximation for the problem of maximizing the number of edges across partitions.