CS 7910 Computational Complexity
Final Exam

Gopal Menon

May 4, 2016

1. **(10 points)** What is NP? What is $NP-Complete$?

   NP stands for Non-Deterministic Polynomial. NP is the set of all problems for which algorithms run in polynomial time on a non-deterministic machine. This machine can be thought of one that can take all possible paths in parallel. On a real machine with finite resources, all NP algorithms will not run in polynomial time (unless if $P = NP$, which is highly unlikely). The set of problems for which algorithms run in polynomial time is called P and $P \subseteq NP$. Another property of NP algorithms is that even though they cannot all run in polynomial time, given a certificate that is a solution to an NP problem, it can be verified in polynomial time that the certificate is correct.

   $NP-Complete$ is the set of the hardest problems in NP. All NP problems can be reduced to $NP-Complete$ problems in polynomial time (as long as the certificate can be verified in polynomial time). Actually if an NP problem can be reduced to another problem, then the resulting problem is said to be in the set $NP-Hard$. The intersection of the sets NP and $NP-Hard$ is the set $NP-Complete$.

2. **(15 points)** For each of the following statements, please tell whether it is true and briefly explain your answer.

   a) There is no polynomial-time algorithm that can solve the vertex cover problem.

   This is true. The Vertex Cover problem is in $NP-Complete$. This means that any problem in NP can be reduced to a Vertex Cover problem in polynomial time. As a result the Vertex Cover problem is among the hardest problems in NP. If there exists a polynomial-time algorithm that can solve the Vertex Cover problem, then any problem in NP (including $NP-Complete$ problems) can be solved in polynomial time by first reducing it to the Vertex Cover problem and then solving the Vertex Cover problem in polynomial time. This would mean that $P = NP$. Since this is very unlikely, then it is safe to say that there is no polynomial-time algorithm that can solve the Vertex Cover problem.

b) If there exists a polynomial-time algorithm that can solve the vertex cover problem, then there exists a polynomial-time algorithm that can solve the load balancing problem.

This is true. Both the Vertex Cover and Load Balancing problem (where jobs need to be efficiently assigned to machines) are in $NP-Complete$. This means that the Load Balancing problem can be reduced to the Vertex Cover problem in polynomial time. If there exists a polynomial-time algorithm that can solve the Vertex Cover problem, then the Load Balancing problem can be solved in polynomial time by first reducing it to an instance of the Vertex Cover problem and then solving the Vertex Cover problem in polynomial time.

c) If there does not exist a polynomial-time algorithm that can solve the vertex cover problem, then there does not exist a polynomial-time algorithm that can solve the load balancing problem.

This is true. The Vertex Cover problem can be reduced to the Load Balancing problem in polynomial time since they are both in $NP-Complete$. This means that the Load Balancing problem is at least as hard as the Vertex Cover problem. If there does not exist a polynomial-time algorithm that can solve the Vertex Cover problem, then a polynomial time algorithm to solve the Load Balancing problem cannot exist since it is at least as hard as the Vertex Cover problem.

d) Quasi-polynomial, weakly-polynomial, and pseudo-polynomial are all polynomial.

This is false.

i. Quasi-polynomial (QP) run time is greater than polynomial time and less than exponential time - $n^d < QP < 2^n$, for a constant $d \geq 0$, where n is the input size. So this is not polynomial time.

ii. Weakly-polynomial running time is a function of the number of input bits. For example $O(n \cdot logM)$. Here M is the input value and $logM$ is the number of input bits. So this is not a polynomial function of the input size n.

iii. Pseudo-polynomial time is a polynomial function of the input value (as opposed to input size). An example is the Dynamic Programming solution for a Subset Sum problem which is $O(n \cdot M)$, where n is the input size and M is the input value for the target sum. This can be shown to be of the order of $O(n \cdot 2^m)$, where $m = logM$. So if we need to find the subset sum of 100 bit integers, we need to have a table if size $n \cdot 2^{100}$, in order to find a subset sum for a target value that uses the 100 bits. So this is actually exponential in nature.

iv. Polynomial time, or strongly polynomial time is where the run time is a function of the input size - $O(n^d)$, where $d \geq 0$.

e) Suppose A is an approximation algorithm for the vertex cover problem and the approximation ratio of A is 2; then it is also correct to say that the approximation ratio of A is 3.

Since A is an approximation algorithm for the Vertex Cover problem, which is a minimization problem, then $\frac{C}{OPT} \leq 2$, where C is the cost of the approximation

algorithm and OPT is the cost of the optimal solution. If $\frac{C}{OPT} \leq 2$, then it is also true that $\frac{C}{OPT} \leq 3$. So it is also correct to say that the approximation ratio of A is 3.

3. **(10 points)** Let G be a bipartite graph, i.e., its nodes are partitioned into two sets X and Y so that each edge has one end in X and the other in Y. We define an $(a,b)-skeleton$ of G to be a subset E' of edges of G so that *at most* a nodes in X are incident to an edge in E' and *at least* b nodes in Y are incident to an edge in E'. Refer to Figure 1 for an example.

   Given a and b, the *skeleton problem* on G is to decide whether G has an $(a,b)-skeleton$. Prove that the skeleton problem is NP-Complete.

   We can try to reduce the Vertex Cover problem, which is in NP-Complete, to the skeleton problem.

   Let the instance of the Vertex Cover problem be (H,k), where H is a graph and we need to find a Vertex Cover consisting of at most k vertices. Let (X,Y,a,b) be the instance of the skeleton problem where X,Y,a,b are as described above.

   To construct the instance of the skeleton problem from the vertex cover problem, do the following:

   a) Make X the set of all vertices in H

   b) Take the set of all edges in H. For each edge in H, construct a vertex in Y and two edges from that vertex in Y to the two vertices in X that are on the two ends of the edge.

   c) Set $a = k$

   d) Set b = number of vertices in Y or the number of edges in H. Both are the same value

   Clearly this construction can be done in polynomial time.

   Given an instance of the skeleton problem (X,Y,a,b) and a certificate, which will be the set of vertices in X and Y that are the solution, it can be verified in polynomial time that the number of vertices in X is at most a and the number of vertices in Y is at least b. We can also verify in polynomial time that the set of nodes in X that are part of the certificate, are incident on the nodes in Y that are part of the certificate. So the skeleton problem is in NP.

   Consider the case where there exists a vertex cover that is of size at most k elements. This means that there will be at most a vertices or nodes in X that cover (or are incident to) all the vertices or nodes in Y. This means that the nodes in X cover at least b nodes in Y.

   In the case where there exists an (a, b) skeleton of the bipartite graph, the there will exist a vertex cover consisting of at most k nodes (since $a = k$) that cover all the edges in the graph.

   This means that we can reduce the vertex cover problem in polynomial time to the skeleton problem. So the skeleton problem is in NP-Hard. Since the skeleton problem is also in NP, we can conclude that the skeleton problem is in NP-Complete.

4. **(5 points)** What is approximation ratio?

   An algorithm is said to have an approximation ratio of $r(n)$ if for an input of size n, the cost C of the solution produced by the algorithm is within a factor $r(n)$ of the cost

OPT of the optimal solution. For a problem that is looking for a minimization solution $\frac{C}{OPT} \leq r(n)$. And for a problem that is looking for a maximization solution $\frac{OPT}{C} \leq r(n)$. We can combine the two and say $max\left(\frac{C}{OPT}, \frac{OPT}{C}\right) \leq r(n)$.

In simple terms we can say that the worst solution provided by an approximate algorithm is limited by the approximation ratio.

5. **(10 points)** Consider the dual load balancing problem in Question 2 of Assignment 10. In Question 2(c), based on the assumption that $t_i \leq \frac{A}{2m}$ for each job i, we have shown that the simple greedy algorithm (i.e., consider the jobs in an arbitrary order and assign each job to the machine with the minimum load) can give a 2-approximation solution.

   Suppose we do not have the above assumption that $t_i \leq \frac{A}{2m}$. Does the simple greedy algorithm still give a 2-approximation solution? Please justify your answer (i.e., if yes, then give the proof; otherwise, give a counterexample).

   The simple greedy algorithm will not give a 2-approximation solution when we do not have the above assumption that $t_i \leq \frac{A}{2m}$.

   Consider the case where the jobs sizes arrive as per the order $\{2,2,2,2,2,8,8,8,8\}$, and there are 5 machines. As per the greedy algorithm, the job sizes assigned to the machines will be

   $$\{\{2,8\},\{2,8\},\{2,8\},\{2,8\},\{2\}\}$$

   The minimum machine load as per the greedy algorithm will be 2. The optimal job assignment to the machines will be

   $$\{\{8\},\{8\},\{8\},\{8\},\{2,2,2,2,2\}\}$$

   So $C = 2$ and $OPT = 8$. The approximation ration will be $\frac{OPT}{C} = \frac{8}{2} = 4$.

6. **(10 points)** Given a set of k clauses $C_1, C_2, ..., C_k$ (each having three distinct literals) over a set of n variables $x_1, x_2, ..., x_n$, the MAX-3SAT problem is to compute an assignment for the variables such that the number of satisfied clauses is maximized.

   In class we studied a randomized algorithm for the MAX-3SAT problem. We have shown that by assigning each variable x_i to 1 with probability 0.5, the expected number of satisfied clauses is $\frac{7}{8} \cdot k$. Using this result, prove that if $k \leq 7$, then there always exists an assignment for the variables that satisfies all clauses.

   The expected number of satisfied clauses is $\frac{7}{8} \cdot k$. For $k \leq 7$, the expected number of satisfied clauses is a real number with a floor of $k-1$ and a ceiling of k. Since this is the expected or average number of satisfied clauses, and the actual number of satisfied clauses has to be an integer, this means that for some assignments, the number of satisfied clauses has to be k. Since k is the total number of clauses, we can confidently say that there always exists an assignment for the variables that satisfies all clauses when $k \leq 7$.