Sample Solution to Exam in Computational Complexity

2007/9/3

Oded Regev, Oded Schwartz, Amnon Ta-Shma

Disclaimer: Although we tried to be careful, these solutions might still contain a bug or two. If you find any, please let us know!

1. Notice that the 1VC problem can be equivalently formulated as asking for a subset U such that every edge has one vertex in U and one vertex in $V \setminus U$. But this is equivalent to asking whether the graph is bipartite, and as we have seen in class, the problem of checking whether a graph is bipartite is in NL.

   To recall, the proof goes as follows. Since NL = coNL, it is enough to show that the problem of checking whether a graph is non-bipartite is in NL. Since this is equivalent to the existence of an odd-cycle, we can easily construct an NL verifier that expects to get as a witness a list of vertices that form a cycle of odd lengths. Verifying this witness is easily done in log-space.

2. (a) We show a reduction from VC, which is known to be NP-hard. Let $(G = (V, E), k)$ be an instance of VC, i.e., our goal is to decide if the undirected graph $G = (V, E)$ contains a vertex cover of size k. We output an instance of DH3S with $|V| + |E|$ items, $|E|$ sets of size 3 each, and the number k. We think of the $|V| + |E|$ items as corresponding to the set $V \cup E$; hence there are $|V|$ 'vertex items' and $|E|$ 'edge items'. For each edge in E we add a 3-set containing the two items corresponding to the vertices touching the edge, and the item corresponding to the edge itself.

   We now prove correctness. Assume that G contains a vertex cover of size k. Then, consider the set A of all items corresponding to the vertices of the vertex cover. Then A intersects all $|E|$ subsets (since one of the two 'vertex items' in each subset must be in the vertex cover), and is of size k. Conversely, assume that there exists a set A of size at most k that intersects all $|E|$ subsets. Observe that we can assume without loss of generality that A does not contain any of the 'edge items': if it does, we can remove this edge item and instead put one of two vertex items of that edge. This does not increase the size of A, and still maintains the property that A intersects all subsets (since each edge item appears only in one subset). To complete the proof, notice that since A contains only 'vertex items', these vertices must form a vertex cover in G whose size is at most k.

   (b) Recall that VC is known to be NP-hard to approximate to within some constant $c > 1$. I.e., it is NP-hard to decide whether the size of the minimum vertex cover is at most k or more than ck. Now by applying the reduction of Item (a), we see that it is NP-hard to decide if the size of the minimum hitting set is at most k or more than ck.

   (c) The algorithm is very similar to the 2-approximation algorithm for VC. It starts by constructing a maximal family of non-intersecting subsets: i.e., as long as there exists a subsets that does not intersect any of the previously chosen subsets, add it to the family. The algorithm then outputs all the elements contained in the chosen subsets.

   The running time is clearly polynomial. Moreover, by construction, all subsets must intersect the output set. To show that we obtain a 3-approximation ratio, notice that an

optimal solution must contain at least one element from each of the subsets we added to our family.

(d) Let A be an algorithm that approximates vertex cover to within 1.8 on triangle-free graphs. We now construct an algorithm that achieves the same approximation ratio for general graphs. Given a graph $G = (V, E)$, we start by finding a maximal set of vertex-disjoint triangles $S \subseteq V$ (i.e., as long as $V \setminus S$ contains a triangle $\{i, j, k\}$, add these three vertices to S). We then run A on the subgraph induced on $V \setminus S$ and obtain some vertex cover $T \subseteq V \setminus S$. Our algorithm then returns $T \cup S$.

First, we notice that the algorithm is correct since any edge in the graph must either touch S, or otherwise appear in the subgraph induced on $V \setminus S$, and therefore be covered by T by the correctness of A. We now analyze the approximation ratio. Let OPT denote the optimal vertex cover of G. Notice that $|\mathrm{OPT} \cap S| \geq 2|S|/3$ since any solution must contain at least two vertices of each triangle. Moreover, by our assumption on A and since $\mathrm{OPT} \cap (V \setminus S)$ is a vertex cover of the subgraph induced on $V \setminus S$, $|T| \leq 1.8 \cdot |\mathrm{OPT} \cap (V \setminus S)|$. Hence,

$$|T \cup S| \leq 1.8 \cdot |\mathrm{OPT} \cap (V \setminus S)| + 1.5 \cdot |\mathrm{OPT} \cap S| \leq 1.8 \cdot (|\mathrm{OPT} \cap (V \setminus S)| + |\mathrm{OPT} \cap S|) = 1.8|\mathrm{OPT}|.$$

3. **Note:** Many students incorrectly assumed that $2^{O(\mathrm{poly}\log n)}$ is polynomial in n. This is obviously wrong; for instance, $2^{\log^2 n} = n^{\log n}$ which is *not* polynomial in n.

(a) True. It is enough to show $\mathsf{SC} \subseteq \mathsf{coSC}$. Let $L \in \mathsf{SC}$ and let M be a poly-time polylog-space deterministic machine that decides L. Then the machine M' that is obtained from M by inverting its output, is a poly-time polylog-space deterministic machine that decides $\bar{L}$. This shows that $\mathsf{SC} \subseteq \mathsf{coSC}$.

(b) False. Savitch's theorem shows that PATH can be solved by a deterministic machine running in space $O((\log n)^2)$. The running time of that machine, however, is $n^{O(\log n)}$, and this therefore is not enough to show that $\mathsf{PATH} \in \mathsf{SC}$. (In fact, it is still an open question whether $\mathsf{PATH} \in \mathsf{SC}$.)

(c) True. Let $B \in \mathsf{SC}$ and let M be a poly-time polylog-space machine that solves B. Assume that there exists a log-space reduction f from A to B. Consider the machine M' that runs M on the 'implicit' input $f(x)$, in the same way that we have shown in class; i.e., each time M needs to look at a bit in $f(x)$, we compute $f(x)$ from scratch, without storing any of the output bits except the required one.

Clearly M' solves A. Moreover, its running time is the product of the running time of f on input x (which is polynomial, since f is a log-space reduction) and the running time of M on an input of size $|f(x)|$ (which is also polynomial in $|x|$ since $|f(x)|$ is polynomial in $|x|$). Finally, its space requirement is poly-logarithmic in $|x|$ since it is the sum of the following things: space requirement of f (which is logarithmic in $|x|$), the space requirement of M on input of size $|f(x)|$ (which is still poly-logarithmic in $|x|$), and finally, $2 \log |f(x)|$ space for the two counters (used to emulate the output tape of f and the input tape of M).

4. (a) True. Let L be any language in P, and let M be a polynomial-time deterministic Turing machine that decides L. Our goal is to show that $L \in \mathsf{UP}$. For that, choose the polynomial $p(|x|) = 1$, and consider the deterministic Turing machine M' that on input x and $y \in \{0,1\}$ runs $M(x)$ and outputs TRUE if M returns TRUE and $y = 1$, and outputs FALSE otherwise. Clearly M' runs in polynomial time. Moreover, if $x \notin L$ then M' always outputs FALSE, no matter what y is, because $M(x)$ returns FALSE. If, on the other hand, $x \in L$, then there is exactly one witness $y = 1$ for which $M'(x, y) =$TRUE, as required.

   (b) True. Let L be any language in UP, and let p be a polynomial and M be a poly-time deterministic Turing machine as in the definition of UP. By using the witness-based definition of NP we see immediately that $L \in \mathsf{NP}$: indeed, the machine M is itself already a poly-time deterministic verifier for L (if $x \in L$ then there is a witness $y \in \{0,1\}^{p(|x|)}$ for which $M(x, y) =$TRUE whereas if $x \notin L$ then for all witnesses $y \in \{0,1\}^{p(|x|)}$, $M(x, y) =$FALSE).

   (c) Let us take the oracle $O = A$ and the language $L = U_A$ where A and U_A are as in the proof of the theorem by Baker, Gill, and Solovay. We already showed in class that $U_A \notin \mathsf{P}^A$ and so it is enough to show that $U_A \in \mathsf{UP}^A$. To recall, we defined

$$U_A = \{1^n \mid \text{some string of length } n \text{ is in } A\}.$$

   Moreover, our construction of A was such that for each n there is at most one string $y \in \{0,1\}^n$ that is in A (i.e., there is at most one needle in each haystack).

   In order to show that $U_A \in \mathsf{UP}^A$ we use exactly the same verifier M as was used to show that $U_A \in \mathsf{NP}^A$. Namely, given an input $x \in \{0,1\}^n$ and a witness $y \in \{0,1\}^n$, M first checks that x is all ones. If not, it rejects immediately. Otherwise, it queries the oracle A with y, and accepts if and only if the oracle returns 1. To prove correctness, notice that if $x \notin U_A$ then there is no string of length n in A, so the oracle must return 0 for each query y. Moreover, if $x \in U_A$ then there is exactly one y that makes $M(x, y)$ accept, namely, the only $y \in \{0,1\}^n$ that is in A.