CSE200: Computability and complexity
Homework set 2 solution

Shachar Lovett

1 Primality in NP

To prove that a number n is prime, the proof is composed of:

1. Numbers $q_1, \ldots, q_r$ such that $n - 1 = \prod_{i=1}^{r} q_i$.
2. Number $a \in \{2, 3, \ldots, n-1\}$ such that $a^{n-1} = 1 \pmod{n}$ and $a^{(n-1)/q} \neq 1 \pmod{n}$.
3. Recursive proof that each of the q_i is prime.

We need to verify a few things: that such a proof exists iff n is prime; that the proof length is polynomial in the length of the description of n, e.g. in $\log(n)$; and that the proof can be verified in polynomial time. The first claim follows by induction on n. By induction $q_1,\ldots,q_r$ are all primes, and all the other facts can be verified in polynomial time easily (recall that we showed in class that powers $a^b \pmod{n}$ can be computed in time $\mathrm{poly}\log(n)$ using repeated squaring). The only thing we should actually verify is that the proof length is indeed polynomial in $\log(n)$ - given that it is obvious is can be verified in $\mathrm{poly}\log(n)$ time.

Let $L(n)$ denote the proof length for n. Then

$$L(n) \leq r\log(n) + \log(n) + \sum_{i=1}^{r} L(q_i) \leq \log^2(n) + \sum_{i=1}^{r} L(q_i).$$

We will prove that $L(n) \leq 2\log^2(n)$ by induction on n. Note that $q_i \leq \sqrt{n}$ and hence $\log(q_i) \leq \log(n)/2$. Hence

$$\begin{aligned} L(n) &\leq \log^2(n) + \sum_{i=1}^{r} L(q_i) \leq \log^2(n) + 2\sum_{i=1}^{r} \log^2(q_i) \\ &\leq \log^2(n) + \log(n)\sum_{i=1}^{r} \log(q_i) = \log^2(n) + \log(n)\log(n-1) \leq 2\log^2(n). \end{aligned}$$

2 Graph coloring

(i) In order to show that 3-coloring is NP-hard we will reduce $3SAT$ to it. This is a classic reduction, and a nice exposition of it can be found at `http://www.ugrad.cs.ubc.ca/~cs320/2010W1/handouts/3color.html`.

(ii) It is clearly enough to show that any connected component of a graph G is 2-colorable. In a connected component C, let v be an arbitrary vertex and color it red. As long as there are uncolored vertices in C, do the following: for any $v \in C$, if v is colored red then color its uncolored neighbors blue; and if v is colored blue color its uncolored neighbors red. If there is an edge with two endpoints colored the same the graph is not 2-colorable, and otherwise this will produce a 2-coloring of C. Note that this coloring of C is unique up to renaming the colors. In an intuitive level, the reason 2-coloring is easy is because given a color of a vertex there is a unique way to color its neighbors in a consistent way.

(iii) In order to implement this algorithm in logspace, we would need several subroutines. We will assume access to a function $CONNECTED(s,t)$ which returns 1 iff s,t are in the same connected component.

(a) We need a way to find a canonical "first vertex" in each connected component. Lets assume the vertices are $V = \{1,2,\ldots,n\}$ and define the representative of a connected component as the minimal vertex in it. Then we can define $REPRESENTATIVE(s)$ to be the canonical representative of the connected component to which a vertex s belongs. Note that we can compute $REPRESENTATIVE(s)$ in logspace by enumerating all vertices in G by order, and outputting the first one which is connected to s.

(b) We next need to color a vertex s based on its distance from $r = REPRESNTATIVE(s)$. Lets define this color to be red if there is an even length path from r to s, and otherwise we color it blue. To do that, we need to check if there is an even length path from r to s. Define the graph G^2 to be a graph on V, where $(u,v) \in G^2$ iff there is a path of length 2 in G between u and v. Note that we can compute the query $(u,v) \in G^2$ in logspace given access to G, by enumerating over all possible vertices w and checking if both $(u,w),(w,v) \in G$. Then, running $CONNECTED$ using the graph G^2 will answer if there is an even length path between two vertices. Hence, we can compute $COLOR(s)$ in logspace.

(c) Finally, we need to verify that the colors of any two adjacent vertices are different. This can clearly by done in logspace.

(iv) Here is a simple algorithm which colors a 3-colorable graph on n vertices with $\sqrt{n}$ colors: if there exists a vertex v with $\sqrt{n}$ neighbors, then this neighbors set is 2-colorable. Color it with two new colors and discard these vertices. This can occur at most $\sqrt{n}$ times and hence uses at most $2\sqrt{n}$ colors. If all vertices have degree less than

$\sqrt{n}$ then the graph can be colored greedily with $\sqrt{n}$ colors: go over the vertices one by one, and for each one assign the minimal color not already assigned to any of its colored neighbors. The number of colors this requires is at most the maximal degree of the graph plus one, hence also $\sqrt{n}$ colors. There are better algorithms which achieve $O(n^c)$ colors for $c \approx 0.2$. It is unknown whether for any $\varepsilon > 0$ there exists a poly-time algorithm which achieves n^ε colors.

3 Size blowup

Assume that $P = NP$. Let $L \in NEXP$ be a language decidable by a nondeterministic Turing machines in time 2^{n^c}. Equivalently,

$$x \in L \iff \exists y, |y| \leq 2^{n^c}, M(x, y) = 1$$

where M is poly-time deterministic machine. Define a new language by padding a sequence of 2^{n^c} ones to any potential input x with $|x| = n$,

$$L' = \{(x, 1^{2^{|x|^c}}) : x \in L\}.$$

We have that $L' \in NP$. By assumption, we also have $L' \in P$. That is, there exists a poly-time machine deciding if $(x, 1^{2^{|x|^c}}) \in L$. This translates into a deterministic machine deciding membership in L in time 2^{n^c}. Hence $L \in EXP$.

4 Union and intersection of NP

(i) Let $L_1, L_2 \in NP$. This means there exists poly-time Turing machines M_1, M_2 such that for $i = 1, 2$,

$$x \in L_i \iff \exists y, |y| \leq |x|^c, M_i(x, y) = 1.$$

We thus have

$$x \in L_1 \cup L_2 \iff \exists y_1, y_2, |y_1|, |y_2| \leq |x|^c, M_1(x, y) = 1 \vee M_2(x, y) = 1$$

and

$$x \in L_1 \cap L_2 \iff \exists y_1, y_2, |y_1|, |y_2| \leq |x|^c, M_1(x, y) = 1 \wedge M_2(x, y) = 1.$$

Hence both $L_1 \cup L_2, L_1 \cap L_2 \in NP$.

(ii) In general, if L_1, L_2 are NP-complete then their union or intersection might be trivial (in particular in P). Of course, if $L_1 = L_2$ then the union/intersection is also NP-complete, showing that there is no general statement about hardness of unions/intersections of NP-hard problems. Now for the examples. Let L be some NP-complete problem and set $L_1 = \{1x : x \in L\}$ and $L_2 = \{0x : x \in L\}$. Clearly both L_1, L_2 are NP-complete

but $L_1 \cap L_2 = \emptyset$ which is trivial. For the other case, let $L_1 = \{1x : x \in L\} \cup \{0x : x \in \{0,1\}^*\}$ and $L_2 = \{0x : x \in L\} \cup \{1x : x \in \{0,1\}^*\}$. Then still L_1, L_2 are NP-complete but $L_1 \cup L_2 = \{0,1\}^*$ which again is trivial.

Interesting enough, it is an open problem whether the union of two **disjoint** NP-complete problems is NP-complete.