Problem Set 5 Complexity

Bayard Walsh

February 2024

1

1.1 a

Given a polynomial-time mapping reduction f from L_1 to L_2, and $L_2 \in QP$.

Claim: if $L_2 \in QP$, then $L_1 \in QP$

Given some $w \in \Sigma_1^*$

Compute $f(w) \in \Sigma_2^*$. This takes $|w|^{k_1}$ time as f is a poly-time mapping reduction

Check whether $f(w) \in L_2$. This takes $2^{\log^{k_2}(|f(w)|)}$ time as $L_2 \in QP$

Accept if $f(w) \in L_2$, reject if $f(w) \notin L_2$

$|f(w)| \leq |w|^{k_1}$, so total time is at most $|w|^{k_1} + 2^{\log_2^{k_2}(|w|^{k_1})}$

$|w|^{k_1} \leq 2^{\log_2^{k_3}(|w|^{k_1})}$ as $P \subseteq QP$ (pset 4)

$2^{\log_2^{k_3}(|w|^{k_1})} + 2^{\log_2^{k_2}(|w|^{k_1})} = 2^{\log_2^{O(1)}(|w|)}$

As we have runtime $2^{\log_2^{O(1)}(|w|)}$ for L_1, $L_1 \in QP$

1.2 b

Let us have some arbitrary L' such that $L' \in EXP$ and $L' \notin QP$. We have that $QP \subseteq EXP$, and $QP \neq EXP$ (pset 4). Therefore we have that this arbitrary L' exists.

L is EXP complete as given. Therefore some polytime mapping f exists for every arbitrary $L'' \in EXP$ such that $f(L'')$ reduces to L.

As $L' \in EXP$, $f(L')$ reduces to L.

We apply the contra positive of $1a$), and because $L' \notin QP$ and $f(L')$ reduces to L, $L \notin QP$ if L is EXP complete.

2

2.1 a

We write a reduction from HALT to HALT-LENGTH to show that HALT-LENGTH is undecidable and therefore $\notin P$.

Let f be some function such that $f(\langle M, w\rangle) = \theta$
Simulate M on w
if M halts on w then $\langle M, w\rangle \in$ HALT
let x' be the string encoding of $\langle M, w\rangle$
x' is also $\in$ HALT
let $x' = \langle |x| \rangle_q$, where x is all strings of length x' in base q
now find θ instance in the set x
let θ be a string of ones with length x'' where x'' is the base q conversion of x' from the HALT alphabet. Therefore $\theta = \{1\}^{x''}$

Yes to Yes
If M halts on w, then we can construct θ so that $\theta \in$ HALT-LENGTH following the algorithm.

No to No
Using the contra positive, if $\theta \in$ HALT-LENGTH, then M must halt on w. If $\theta \in$ HALT-LENGTH then we can convert θ to $\langle M, w\rangle$ by following the opposite direction of the algorithm, and setting $\langle |\theta| \rangle_q = \langle M, w\rangle$ which corresponds to a $\langle M, w\rangle$ instance of HALT by HALT-LENGTH definition. Therefore if M doesn't halt on w then $\theta \notin$ HALT-LENGTH.

Computable
In f we can compute x', x'', θ through constant time operations, so f is computable.

Therefore HALT-LENGTH is undecidable and $\notin P$.

2.2 b

To show HALT-LENGTH $\in$ PSIZE, we need to prove some polynomial circuit family $C = \{c_0, c_1, \cdots, c_n\}$ decides HALT-LENGTH

For every input $x \in \{0,1\}^*$ let $|x| = l$. Then we construct circuit c_l for inputs of length l such that $c_l(x)$ returns 1 if $\langle l \rangle_q \in$ HALT and 0 if $\langle l \rangle_q \notin$ HALT. These circuits are constructable through first converting the input x to its base q representation and then checking for membership in HALT.

Therefore if $x \in$ HALT-LENGTH and $|x| = l$, $c_l(x) = 1$. Furthermore, if $c_l(x) = 1$, then x must be in HALT-LENGTH as $\langle l \rangle_q \in$ HALT and $|x| = l$. Therefore any arbitray length x can be decided through an arbitray $c_l(x)$, so taking all $\{c_0, c_1, \cdots, c_n\}$ as C we have that circuit family C decides HALT-LENGTH.

By definition of a circuit family, we can create circuits that handle every specific sized input for HALT-LENGTH. We also know that a given circuit c_i has polynomial size because the work it does to convert to base q and check for $\langle l \rangle_q$ membership $\in$ HALT is polynomial, meaning the size of a given circuit has a polynomial bound relative to the size of input x. Therefore, as all circuits $\in C$ are poly-sized with respect to the input x, we have a polynomial sized circuit family that can solve HALT-LENGTH, meaning HALT-LENGTH $\in$ PSIZE.

3

3.1 a

Certificate
Let the $k-$colorable certificate be a string of the following pattern:
$CERT = \{1(v_1, \cdots, v_i), 2(v_2, \cdots, v_{i+1}), \cdots, k(v_3, \cdots, v_{i+2})\}$

The specific placements of v_i are not important here, but $CERT$ is an example of transforming a ϕ assignment into a string, which assigns each v to a given color through passing a list of color labels and their corresponding nodes in $CERT$.

Each node can only occur once in $CERT$, as each node can only be assigned one color in a valid $k-$colorable partition. Each color label also only occurs once, in front of the nodes it labels. For each coloring, there are additional characters "()," for parsing purposes, and each node has a comma after it to distinguish it from the previous node, but these characters are constant relative to the size of V and k. Therefore these characters are $O(1)$ length for a given v, and the size of $CERT = V + K$, as $CERT$ will only list each k, v once. We cannot have more labels than nodes in a valid certificate (trivially we can do a 1 to 1 label to node assignment for a valid $k-$ coloring if $V < K$), so $K \leq V$, and

$|CERT| = 2V = V$. Therefore $|CERT| \leq |V|^k$, meaning our certificate has polynomial size.

Let R be the **Verification Algorithm** described below, which is given inputs $\langle G, CERT \rangle$

Consider if G is $k-$colorable. Therefore, there must be some ϕ such that $\phi(u) \neq \phi(v)$ for every edge $\{u, v\} \in E$. If a valid ϕ exists, we can convert a given ϕ to a valid $CERT$ by checking every node $n \in G$ and constructing $CERT$ through querying $n, \phi(n)$. As shown above $|CERT| \leq |V|^k$. Therefore if $G \in k-$colorable there exists string $CERT$ such that $|CERT| \leq |w|^k$ and $\langle G, CERT \rangle \in R$

Consider if G is not $k-$colorable. For contradiction, assume that some $CERT$ exists such that $\langle G, CERT \rangle \in R$. Therefore $CERT$ contains the list representation of a valid way of labeling nodes in G to form a $k-$coloring. Let us construct a valid ϕ' through querying through $CERT$, or more formally, if $CERT = \cdots n(\cdots, u, \cdots) \cdots$ then $\phi'(u) = n$ for a given node u. However, if G is not $k-$colorable, then no valid ϕ exists such that $\phi(u) \neq \phi(v)$ for every edge $\{u, v\} \in E$, meaning ϕ' can't be valid and there cannot be any $CERT$ such that $\langle G, CERT \rangle \in R$. Therefore for every string $CERT$ with $|CERT| \leq |w|^k$ we have $\langle G, CERT \rangle \notin R$

Verification Algorithm

Add labels $\delta, Q \in \Sigma$
Given $\langle G = (V, E), CERT \rangle$, pick any arbitrary node $v \in V$.

Label v as δ *(label v as visited)*
Add all neighbors of v, $n_1, n_2 \cdots n_i$ to Q. *(add neighbors to queue)*

For every $q \in Q$:
    remove q from Q
    label q as δ *(label q as visited)*
    For every neighbor $n'_1, n'_2, \cdots n'_k$ of q:
        if n'_j isn't in δ add n'_j to Q

        Get labels for n'_j, q through querying $CERT$
        if n'_j is labeled the same color as q, return False
        if n'_j is labeled a different color from q, continue to next neighbor

if Q is empty return True

k coloring verification

Verification algorithm visits every node $v \in G$ once. For every v, it checks every

neighbor of v. Therefore it checks every edge $\{u, v\}$ and checks if $\phi(u) \neq \phi(v)$ (through querying the $CERT$ encoding). If $\phi(u) = \phi(v)$ once it returns False, and if it runs out of nodes to check its returns True. Therefore assuming $CERT$ is a valid encoding of a valid ϕ, as shown above, our verification algorithm will verify the certificate by checking the ϕ value of every possible edge which determines if a graph is $k-$colorable by definition.

With a total number of nodes $|V|$, each visited node has a maximum of $|V| - 1$ neighbors. *Note that if every node had this amount of neighbors it would not be a valid* $1 < k$ *coloring graph, but it serves as an upper bound for neighbor checking.* For each neighbor, the algorithm does a constant amount of work in terms of labeling visited nodes which is $O(1)$, and in order to check membership for a given node, it takes linear time to query through $CERT$, which has size $|V|$ (as shown previously). Therefore checking membership between two nodes will take $2|V|$ or $|V|$ time. Therefore the algorithm has a runtime of $|V| \cdot (|V| - 1) \cdot |V|$, or $|V|^3$, which is polynomial as $|V|^3 \in |V|^k$. Therefore k coloring verification is $\in P$ given a polynomial certificate $CERT$, meaning that $k-$ coloring is $\in NP$.

3.2 b

Algorithm

Add labels $\delta, C_1, C_2, Q \in \Sigma$

Given a graph $G = (V, E)$, pick any arbitrary node $v \in V$.
Label v as C_1,δ *(label* v *as coloring set* C_1 *and label* v *as visited)*
Label all neighbors, $n_1, n_2 \cdots n_i$ of v as C_2.
Add all neighbors, $n_1, n_2 \cdots n_i$ to Q.

For every $q \in Q$:
    remove q from Q
    mark q as δ *(label* q *as visited)*
    check every neighbor $n'_1, n'_2, \cdots n'_k$ of q. For every neighbor n'_j:
    define "the opposite" of C_1 *as* C_2 *and* C_2 *as* C_1
        if n'_j isn't in δ add n'_j to Q
        if n'_j is labeled the same color as q, return False
        if n'_j is labeled the opposite color as q, continue
        if n'_j is unlabeled, label n'_j as the opposite of q and continue

if Q is empty return True

Correctness
Consider if $G \in$ 2-Colorable. Therefore, there must be some function ϕ such that for every edge $\{u, v\}$, $\phi(u) \neq \phi(v)$. Because $k = 2$, every neighbor of a given node must belong to the opposite set of that node, because if any node belonged to the same set as its neighbor there would be a contradiction to the

2 colorable definition, and there is only one other possible label for each node, which is the opposite label. Therefore if a graph G is 2-Colorable, there must be some way of partitioning the nodes of G such that for every node n, and for every neighbor of n, n_i, $\phi(n) \neq \phi(n_i)$. We apply the observation that all neighbors of a given node must belong to the opposite coloring in our algorithm, as our algorithm visits every node $n \in G$ and assigns every neighbor n_i to the opposite color of n, and accepts after verifying that every node upholds this observation. Therefore, if a labeling exists our algorithm will eventually find it and return True after checking that that all neighbors of a given node belong to the opposite label for every node.

Consider if $G \notin$ 2-Colorable. Therefore, there cannot be any function ϕ such that for every edge $\{u, v\}$, $\phi(u) \neq \phi(v)$. Therefore there must be some $\{u', v'\}$ where $\phi(u') = \phi(v')$ for every ϕ. As our algorithm checks every $\{u, v\}$ pair in the graph and attempts to assign neighbors the opposite color of a given node, and returns False when this is not possible, it will eventually find some instance $\{u', v'\}$ where $\phi(u') = \phi(v')$ and return False.

Runtime
Our algorithm visits every node $v \in V$ once. For each visited node, it checks every neighbor. With a total number of nodes $|V|$, each visited node has a maximum of $|V|-1$ neighbors. *Note that if every node had this amount of neighbors it would not be a valid* 2 *coloring graph, but the observation serves as an upper bound for runtime.* For each neighbor, the algorithm does a constant amount of work in terms of labeling and checking membership in visited sets. All of this work is not dependent on V, so we can consider it to be $O(1)$ for a given v. Therefore the algorithm has a runtime of $|V| \cdot (|V| - 1) \cdot O(1)$, or $|V|^2$, which is polynomial. Therefore 2 coloring is $\in P$.

4

Algorithm
Given some arbitrary satisfiable circuit $\langle C \rangle$ with input length $x_1, x_2, \cdots x_n$:
Let $S = \{\}$

Note: define $S + 1$ *as a binary string* S *with a* 1 *added to the end. define* $C' = C$ *with binary string* $S + 1$ *loaded into the front nodes, so* C' *is circuit* C *where inputs* $x_1, x_2, \cdots x_j = S + 1$ *with open inputs* $x_{j+1}, x_{j+2}, \cdots x_n$

For every $x_j \in x_1, x_2, \cdots, x_n$
Load $S + 1$ into C, to get C'

Check if C' is satisfiable using a polynomial CIRCUIT-SAT algorithm.
If C' is satisfiable, add 1 to the end of S and continue to x_{j+1}.
If C' is not satisfiable, add 0 to the end of S and continue to x_{j+1}.

Return S

Proof
$\langle C \rangle$ is initially satisfiable by some x as given. C is also satisfiable by some x when S is empty, as C' loaded with $\{\}$ equals C. Consider the first node, x_1 as the base case. Because the input can only be $0, 1$ for x_1 in a binary circuit and given that C is satisfiable on the empty prefix, we can load 1 into C, and check if the circuit C on $x_2, \cdots x_n$ is satisfiable where $x_1 = 1$ for C. If its is satisfiable, we know 1 is a prefix for a satisfiable input to C, and if its not satisfiable, we know that 0 is a prefix for a satisfiable input to C, because some sastisifable input must exist as given, and that input must start with either 0 or 1. Our algorithm tests this satisfiability for x_1, and will choose whichever $0, 1$ is a prefix to a satisfiable input for C

Assume that we have a satisfiable prefix S where $|S| = k$. Then on x_{k+1}, we can load $S+1$ and check if C' is satisfiable, and then pick either 1 or 0 based on which node results in a satisfiable circuit. As we know S is a satisfiable prefix by assumption, then $S+1$ or $S+0$ must be a satisfiable prefix because there are only two options for the binary circuit, and as we can test if $S+1$ is satisfiable and pick $S+0$ if it isn't, we will always pick the satisfiable option for x_{k+1}, and we can apply this process to any arbitrary node. Therefore, for every node we can always pick the option that satisfies C to eventually construct a prefix S output such that $C(S) = 1$.

Runtime
We know CIRCUIT-SAT runs in NP time as proved in class. Therefore, assuming $P = NP$, some polynomial algorithm must exist that solves CIRCUIT-SAT algorithm, so we have runtime $O(n^k)$ for each CIRCUIT-SAT call. As we apply the CIRCUIT-SAT algorithm once for every input node in the circuit, the overall runtime is $O(n^{k+1})$, meaning our algorithm is polynomial.