

CS 6520: Computational Complexity

Problem Set 1 Solutions

Problem 1

Give a Karp reduction from CLIQUE to SAT.

Solution: Let G be a given graph, with $V(G) = [n]$, and let $k \in [n]$ be a given integer. Define $f(G, k)$ as the following Boolean formula:

1. For each $i \in [k]$ and $j \in [n]$, introduce a variable $X_{i,j}$. The intention is that $X_{i,j}$ is set to 1 if and only if the i -th vertex of the alleged k -clique is vertex j of G .
2. Define

$$f(G, k) \stackrel{\text{def}}{=} \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge \Phi_4,$$

where

- $\Phi_1 \stackrel{\text{def}}{=} \bigwedge_{i \in [k]} \left(\bigvee_{j \in [n]} X_{i,j} \right)$ (for every $i \in [k]$, the i -th vertex of the clique is some vertex of the graph);
- $\Phi_2 \stackrel{\text{def}}{=} \bigwedge_{i \in [k]} \bigwedge_{u,v \in [n], u < v} (\overline{X_{i,u}} \wedge \overline{X_{i,v}})$ (for every $i \in [k]$, the i -th vertex of the k -clique can be at most one vertex of the graph);
- $\Phi_3 \stackrel{\text{def}}{=} \bigwedge_{v \in [n]} \bigwedge_{i,j \in [k], i < j} (\overline{X_{i,v}} \wedge \overline{X_{j,v}})$ (for each $i \neq j$, no vertex can be both the i -th vertex and the j -th vertex of the k -clique);
- $\Phi_4 \stackrel{\text{def}}{=} \bigwedge_{i,j \in [k], i < j} \left(\bigvee_{\{u,v\} \in E(G)} (X_{i,u} \wedge X_{j,v}) \right)$ (there is an edge between every pair of vertices in the k -clique).

It is clear that G has a k -clique if and only if $\Phi_1, \Phi_2, \Phi_3, \Phi_4$ are all satisfiable, and thus $f(G, k)$ is satisfiable. The size of $f(G, k)$ and the time for computing $f(G, k)$ is $O(k^2 n^2)$. \square

Problem 2

Let QUADRATIC be the problem of deciding whether a given system of quadratic multivariate polynomial equations with integer coefficients has a solution modulo 2. Prove that QUADRATIC is **NP-Complete**.

Proof: Clearly $\text{QUADRATIC} \in \text{NP}$. We show that QUADRATIC is **NP-Hard** by reduction from **Circuit-SAT**.

Let $C(x_1, \dots, x_n)$ be a given Boolean circuit with n inputs and let m be the total number of nodes in C (i.e. the total number of all input nodes and gates). Topologically sort the nodes of C . WLOG assume that the n input nodes appear as the first n nodes in the topological order. Thus the last $m - n$ nodes are all the gates in C , and the m -th (i.e. the last) node is the output gate. For each $i \in [m]$, associate a variable z_i to the i -th node.

We arithmetize the circuit C , and define $f(C)$ as the following system of $m - n + 1$ quadratic equations, where for each $k \in \{n + 1, \dots, m\}$:

- if the k -th node is a NOT gate and its input comes from node i , where $i < k$, introduce the equation

$$z_k = 1 - z_i;$$

- if the k -th node is an AND gate and its input comes from node i and node j , where $i, j < k$, introduce the equation

$$z_k = z_i z_j;$$

- if the k -th node is an OR gate and its input comes from node i and node j , where $i, j < k$, introduce the equation

$$z_k = 1 - (1 - z_i)(1 - z_j).$$

Finally introduce the equation $z_m = 1$. The way to understand the reduction is that for each $k \in [m]$, the variable z_k simulates the k -th node of the circuit C , and first $m - n$ equations of $f(C)$ defined above simulate the circuit C . In particular, z_m simulates the output of C . The last equation $z_m = 1$ enforces the satisfiability of C .

Clearly $f(C)$ can be computed in polynomial time.

Suppose that $C(a_1, \dots, a_n) = 1$ for some assignment $(a_1, \dots, a_n) \in \{0, 1\}^n$ to (x_1, \dots, x_n) . It is not difficult to see from the construction of the equations that assigning $z_i \leftarrow a_i$ for each $i \in [n]$ induces an assignment that satisfies all the equations in $f(C)$.

Conversely, suppose that there is an assignment to (z_1, \dots, z_m) that satisfies all the equations in $f(C)$ modulo 2. Then there is an assignment $(a_1, \dots, a_m) \in \{0, 1\}^m$ to (z_1, \dots, z_m) that satisfies all the equations in $f(C)$.

It is not difficult to see, from the construction of the equations, that this implies $C(a_1, \dots, a_n) = 1$. \square

Problem 3

An **NP** *minimization problem* is defined by an objective function $\text{Obj} : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N} \cup \{0\}$ for which there is a constant c and an algorithm that for every $x, y \in \{0, 1\}^*$, computes $\text{Obj}(x, y)$ in time $O(|x|^c)$. For such an objective function Obj , the corresponding **NP** minimization problem is: Given $x \in \Sigma^*$, find a $y \in \Sigma^*$ such that $\text{Obj}(x, y)$ is minimized. Prove that $\mathbf{P} = \mathbf{NP}$ if and only if every **NP** minimization problem has a polynomial-time algorithm.¹

Proof:

(\Leftarrow) Suppose that every **NP** minimization problem has a polynomial-time algorithm. Let $L \in \mathbf{NP}$, and let V be a polynomial-time verifier for L . Define $\text{Obj}(x, y) \stackrel{\text{def}}{=} 1 - V(x, y)$. The function Obj is polynomial-time computable and thus defines an **NP** minimization problem. Moreover, the minimum of $\text{Obj}(x, \cdot)$ is 0 if $x \in L$, and is 1 if $x \notin L$. Hence the following algorithm decides L in polynomial time:

On input x :

1. Find in polynomial time a solution y such that $\text{Obj}(x, y)$ is minimized.
2. If $\text{Obj}(x, y) = 0$ (i.e. $V(x, y) = 1$), output 1;
if $\text{Obj}(x, y) = 1$ (i.e. $V(x, y) = 0$), output 0.

Therefore $\mathbf{P} = \mathbf{NP}$.

(\Rightarrow) Suppose that $\mathbf{P} = \mathbf{NP}$. Consider any polynomial-time computable objective function Obj and the **NP** minimization problem defined by Obj . We will construct a polynomial-time algorithm for the problem. Given an input x , the algorithm will proceed in two phases: It first finds the minimum value m of $\text{Obj}(x, \cdot)$, then finds a solution y such that $\text{Obj}(x, y) = m$.

¹An **NP** *maximization problem* can be defined similarly, for which an analogous result holds.

Finding the minimum value m of $\text{Obj}(x, \cdot)$: Since $\text{Obj}(x, y)$ can be computed in time $O(|x|^c)$ for some constant c , the length of $\text{Obj}(x, y)$ is at most $O(|x|^c)$; that is, the value of $\text{Obj}(x, y)$ is at most $2^{O(|x|^c)}$. Therefore, by $O(|x|^c)$ queries of the form “Is there a y such that $\text{Obj}(x, y) < v$?” to an **NP**-oracle, one can use binary search to find the minimum value of $\text{Obj}(x, \cdot)$.

Finding a y such that $\text{Obj}(x, y) = m$: Recall that if **P** = **NP**, then all **NP** search problems can be solved in polynomial time. Since Obj is polynomial-time computable, the language

$$\{(x, v) : \exists y \text{ s.t. } \text{Obj}(x, y) = v\}$$

is in **NP**. Therefore, if **P** = **NP**, then given x and the minimum value m of $\text{Obj}(x, \cdot)$ computed in the first phase, one can find in polynomial time a y such that $\text{Obj}(x, y) = m$. \square

Problem 4

Prove that for each $i \in \mathbb{N}$, $\Sigma_i\text{-SAT}$ is Σ_i^P -**Complete**.

Proof: Clearly $\Sigma_i\text{-SAT} \in \Sigma_i^P$. We show that $\Sigma_i\text{-SAT}$ is Σ_i^P -**Hard**.

Let $L \in \Sigma_i^P$. By definition, there is a deterministic polynomial-time Turing machine M such that for every string x ,

$$x \in L \iff \exists y_1 \forall y_2 \cdots Q_i y_i M(x, y_1, \dots, y_i) = 1, \quad (1)$$

where each y_i is bounded by a fixed polynomial in length, and $Q_i = \exists$ if i is odd, and $Q_i = \forall$ if i is even. It follows, as in the proof of the Cook-Levin Theorem, that for each x , one can construct, in $\text{poly}(|x|)$ time, a Boolean formula $\varphi_{M,x}$ such that $M(x, y_1, \dots, y_i) = 1$ if and only if there is a string z such that $\varphi_{M,x}(y_1, \dots, y_i, z) = 1$.

First consider the case where i is odd. By the foregoing discussion,

$$x \in L \iff \exists y_1 \forall y_2 \cdots \exists y_i \exists z \varphi_{M,x}(y_1, \dots, y_i, z) = 1.$$

Note that on the RHS the number of alternations is still i . Hence

$$\exists y_1 \forall y_2 \cdots \exists y_i \exists z \varphi_{M,x}(y_1, \dots, y_i, z)$$

is a $\Sigma_i\text{-SAT}$ instance, and the mapping

$$f(x) = \exists y_1 \forall y_2 \cdots \exists y_i \exists z \varphi_{M,x}(y_1, \dots, y_i, z)$$

is a polynomial-time reduction from L to Σ_i -SAT.

Now consider the case where i is even (so that $Q_i = \forall$). We consider \bar{L} , and restate (1) as follows: There is a deterministic polynomial-time Turing machine \bar{M} such that for every string x ,

$$x \in \bar{L} \iff \forall y_1 \exists y_2 \cdots \exists y_i \bar{M}(x, y_1, \dots, y_i) = 1.$$

As before, we can construct, in $\text{poly}(|x|)$ time, a Boolean formula $\varphi_{\bar{M},x}$ such that $\bar{M}(x, y_1, \dots, y_i) = 1$ if and only if there is a string z such that $\varphi_{\bar{M},x}(y_1, \dots, y_i, z) = 1$. Therefore,

$$x \in \bar{L} \iff \forall y_1 \exists y_2 \cdots \exists y_i \exists z \varphi_{\bar{M},x}(y_1, \dots, y_i, z) = 1.$$

The number of alternations is i . Hence

$$\forall y_1 \exists y_2 \cdots \exists y_i \exists z \varphi_{\bar{M},x}(y_1, \dots, y_i, z)$$

is a Π_i -SAT instance, and the mapping

$$f(x) = \forall y_1 \exists y_2 \cdots \exists y_i \exists z \varphi_{\bar{M},x}(y_1, \dots, y_i, z)$$

is a polynomial-time reduction from \bar{L} to Π_i -SAT, which is also a polynomial-time reduction from L to Σ_i -SAT. \square

Problem 5

Consider the FACTORING problem: Given a natural number N , express N as a product of its prime factors. To date no polynomial-time algorithm for FACTORING is known, and the conjectured hardness of FACTORING has been the basis of several public-key cryptosystems.

1. Prove that if $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$, then FACTORING can be solved by a polynomial-time algorithm.

You may find the following facts useful:

- The Fundamental Theorem of Arithmetics: Every integer greater than one can be decomposed into a product of prime factors, and moreover such a decomposition is *unique* up to the order of the prime factors.
- There is a polynomial-time algorithm for deciding whether a given integer is a prime.

Proof: Let language L_{Fac} consist of pairs of the form (N, x) where $N \in \mathbb{N}, x \in \{0, 1\}^*$, such that $(N, x) \in L_{\text{Fac}}$ if and only if there is a string $y \in \{0, 1\}^*$ such that $x \circ y$ is the binary representation of a prime factor of N , where $x \circ y$ denotes the concatenation of strings x and y . That is, $(N, x) \in L_{\text{Fac}}$ if and only if x is a prefix of a string representing a prime factor of N . We show that:

Claim 1 FACTORING *Cook-reduces to* L_{Fac} .

Claim 2 $L_{\text{Fac}} \in \text{NP} \cap \text{coNP}$.

Hence if $\mathbf{P} = \text{NP} \cap \text{coNP}$, then FACTORING can be solved by a polynomial-time algorithm.

Proof of Claim 1: We describe an algorithm that solves the FACTORING problem in polynomial time, given access to a membership oracle $\mathcal{O}_{L_{\text{Fac}}}$ for L_{Fac} . The following algorithm $A^{\mathcal{O}_{L_{\text{Fac}}}}(N)$ finds one prime factor of N in polynomial time, given access to $\mathcal{O}_{L_{\text{Fac}}}$. One can repeatedly use $A^{\mathcal{O}_{L_{\text{Fac}}}}$ to completely factor N ; that is, compute $p_1 = A^{\mathcal{O}_{L_{\text{Fac}}}}(N)$, then $p_2 = A^{\mathcal{O}_{L_{\text{Fac}}}}(N/p_1), \dots$, until N is completely factored. Since N has at most $\log_2 N$ prime factors, the resulting algorithm is still efficient.

Algorithm $A^{\mathcal{O}_{L_{\text{Fac}}}}$:

On input $N \in \mathbb{N}$ (in the binary representation):

- (a) Test whether N is a prime. If so, output N and halt.
- (b) Let $p \leftarrow \varepsilon$, where ε is the empty string.
- (c) Repeat the following (until p represents a prime factor of N):
 - i. Let $b \leftarrow 0$.
 - ii. If $\mathcal{O}_{L_{\text{Fac}}}(N, p \circ b) = 0$, i.e. if $(N, p \circ b) \notin L_{\text{Fac}}$, let $b \leftarrow 1$.
 - iii. Let $p \leftarrow p \circ b$.
 - iv. Test whether p is the binary representation of a prime number. If so, output p and halt.

We first observe that $A^{\mathcal{O}_{L_{\text{Fac}}}}(N)$ finds one prime factor of N : If N is prime then Phase (a) of the algorithm outputs N . If N is composite, then the main loop in Phase (c) finds a prime factor p of N , one bit at a time. We also observe that the running time of $A^{\mathcal{O}_{L_{\text{Fac}}}}(N)$ is

polynomial in n , which is the number of bits representing N . This is so because primality can be tested in polynomial time, and the number of iteration in the main loop in Phase (c) is at most n . \square

Proof of Claim 2: Note that the complete factorization $(p_1^{e_1}, \dots, p_k^{e_k})$ of N serves both as a certificate of membership for (N, x) if $(N, x) \in L_{\text{Fac}}$, and as a certificate of non-membership for (N, x) if $(N, x) \notin L_{\text{Fac}}$. To verify the membership or non-membership of a pair (N, x) , using $(p_1^{e_1}, \dots, p_k^{e_k})$, simply:

- (a) Verify that p_1, \dots, p_k are all primes.
- (b) Verify that $N = \prod_{i=1}^k p_i^{e_i}$.
- (c) Verify that x appears as a prefix of p_i for at least one i .

The correctness of the verification procedure follows from the unique factorization of integers, and its efficiency follows from that of the primality test. Therefore, $L_{\text{Fac}} \in \mathbf{NP} \cap \mathbf{coNP}$. \square

2. Prove that unless $\mathbf{NP} = \mathbf{coNP}$, FACTORING is *not* **NP-Hard** under Cook reduction. Conclude that unless $\mathbf{NP} = \mathbf{PH}$, FACTORING is *not* **NP-Hard** under Cook reduction.

Proof: If FACTORING were **NP-Hard** under Cook reduction, then the language L_{Fac} defined above would also be **NP-Hard** under Cook reduction, as FACTORING Cook-reduces to L_{Fac} and reductions are transitive.

Claim 3 *If a language in $\mathbf{NP} \cap \mathbf{coNP}$ is **NP-Hard** under Cook reduction, then $\mathbf{NP} = \mathbf{coNP}$.*

Since $L_{\text{Fac}} \in \mathbf{NP} \cap \mathbf{coNP}$, it follows that if FACTORING were **NP-Hard**, then $\mathbf{NP} = \mathbf{coNP}$. The latter implies that $\mathbf{NP} = \mathbf{PH}$.

Proof of Claim 3: Let $L \in \mathbf{NP} \cap \mathbf{coNP}$ and suppose that L is **NP-Hard** under Cook reduction. We show that this implies that $\mathbf{NP} \subseteq \mathbf{NP} \cap \mathbf{coNP}$, and thus $\mathbf{NP} \subseteq \mathbf{coNP}$. The latter holds if and only if $\mathbf{NP} = \mathbf{coNP}$.

Let $\tilde{L} \in \mathbf{NP}$. By assumption \tilde{L} Cook-reduces to L . That is, there is an algorithm $A_T^{\mathcal{O}}$ that decides \tilde{L} in polynomial time given access to an

oracle \mathcal{O}_L for the membership of L . We show that $\tilde{L} \in \mathbf{NP} \cap \mathbf{coNP}$. To do so, we show how to certify the membership and non-membership with respect to \tilde{L} .

Consider any string x and the execution of $A_L^{\mathcal{O}}(x)$. Let q_1, \dots, q_m be the queries made by A to \mathcal{O}_L and let a_1, \dots, a_m be the corresponding answers, i.e. $a_i = \mathcal{O}_L(q_i)$ for each i . Since $L \in \mathbf{NP} \cap \mathbf{coNP}$, each member of L has a certificate of membership and each non-member of L has a certificate of non-membership with respect to L . Let π_1, \dots, π_m be the corresponding certificates of q_1, \dots, q_m respectively with respect to L . It follows that with respect to \tilde{L} , $\{(q_1, a_1, \pi_1), \dots, (q_m, a_m, \pi_m)\}$ serves both as a certificate of membership if $x \in \tilde{L}$, and as a certificate of non-membership if $x \notin \tilde{L}$.

To verify the membership or non-membership of x with respect to \tilde{L} using $\{(q_1, a_1, \pi_1), \dots, (q_m, a_m, \pi_m)\}$, simply:

- (a) Run A on x .
- (b) When A makes the i -th query, verify that it is q_i , and verify that a_i is the correct answer using π_i . If so, give a_i to A ; else reject.
- (c) Run A until it halts, and finally verify based on A 's output.

The verification procedure is clearly correct, and is efficient as A is. \square

Problem 6

A language L is *self-reducible* if there is a polynomial-time oracle machine M such that (a) M^L decides L , and (b) on every input x , M only makes queries of length strictly smaller than $|x|$. For instance, as shown in class, SAT and TQBF are self-reducible. Prove that every self-reducible language is in **PSPACE**.

Proof: Suppose that L is self-reducible, and let M be a polynomial-time oracle machine M that decides L given access to an L -oracle for strings shorter than the input. Consider the following algorithm M' :

On input x :

- Simulate $M(x)$.
- When M makes an oracle query q , recursively compute and answer with $M'(q)$.

Algorithm M' requires space for simulating $M(x)$, which is $\text{poly}(|x|)$, since M runs in polynomial time, and space for running $M'(q)$ for a single q as space can be recycled. Let $S(n)$ denote the worst-case space M' uses on inputs of length n . Since $|q| < |x|$, it follows that for all $n > 1$,

$$\begin{aligned} S(n) &\leq S(n-1) + \text{poly}(n) \\ &\leq n \cdot \text{poly}(n) = \text{poly}(n). \end{aligned}$$

□

Problem 7

1. A directed graph $G = (V, E)$ is *strongly connected* if for every pair of vertices $u, v \in V$, there is a path from u to v in G . Prove that the problem of deciding whether a given directed graph is strongly connected is **NL-Complete**.

Proof: It is clear that $\text{STRONG-CONNECTIVITY} \in \text{NL}$: Simply enumerate all pairs of vertices (u, v) of the given graph, and decide non-deterministically whether there is a path from u to v . To show that $\text{STRONG-CONNECTIVITY}$ is **NL-Hard**, we give a log-space reduction from STCONN to $\text{STRONG-CONNECTIVITY}$.

Given $\langle G, s, t \rangle$, simply add an edge from every vertex $u \notin \{s, t\}$ to s , and an edge from t to every vertex $v \in V(G)$. It is clear that the resulting graph can be constructed in log space, and it is strongly connected if and only if there is a path from s to t in G . □

2. Prove that 2-SAT is **NL-Complete**.

Proof: We first show that $\text{2-SAT} \in \text{NL}$.

First note that a clause $(u \vee v)$ is logically equivalent to each of the expressions $(\neg u \rightarrow v)$ and $(\neg v \rightarrow u)$. Given 2-CNF formula Φ , we construct a directed graph G_Φ as follows: The set of vertices $V(G_\Phi)$ consists of the literals of Φ (i.e. the variables of Φ and their negations). For each clause $(u \vee v)$ in Φ , introduce two edges $(\neg u, v) \in E(G_\Phi)$ and $(\neg v, u) \in E(G_\Phi)$, corresponding to the two logical implications. We note that G_Φ has the following interesting symmetry: for every two vertices $u, v \in V(G_\Phi)$, $(u, v) \in E(G_\Phi)$ if and only if $(\neg v, \neg u) \in E(G_\Phi)$. Clearly, given Φ , the graph G_Φ can be constructed in log space.

In the following, for two vertices $u, v \in V(G_\Phi)$ (i.e. two literals u, v of Φ), the notation $u \rightsquigarrow v$ shall mean that there is a path from vertex u to vertex v in G_Φ .

Claim 4 *A 2-CNF formula Φ is unsatisfiable if and only if for there is some variable x such that $x \rightsquigarrow \neg x$ and $\neg x \rightsquigarrow x$.*

From Claim 4, it follows that $2\text{-UNSAT} \in \mathbf{NL}$, as the condition $x \rightsquigarrow \bar{x}$ and $\bar{x} \rightsquigarrow x$ can be tested in nondeterministic log space.² Since $\mathbf{NL} = \mathbf{coNL}$, it follows that $2\text{-SAT} \in \mathbf{NL}$.

Proof of Claim 4:

(\Leftarrow): Suppose there is a variable x for which $x \rightsquigarrow \neg x$ and $\neg x \rightsquigarrow x$. Suppose that Φ were satisfied by an assignment A . WLOG suppose that $A(x) = 1$ (and thus $A(\neg x) = 0$). Since there is a path $x \rightsquigarrow \neg x$, $A(x) = 1$ and $A(\neg x) = 0$, there is an edge (u, v) on this path such that $A(u) = 1$ and $A(v) = 0$. However, by the construction of G_Φ , since $(u, v) \in E(G_\Phi)$, the clause $(\neg u, v)$ is in Φ . But this clause is unsatisfied by the assignment A , a contradiction.

In the case where $A(x) = 0$, we can reach a similar contradiction by considering a path $\neg x \rightsquigarrow x$.

(\Rightarrow): Suppose that for each variable x , $x \not\rightsquigarrow \neg x$ or $\neg x \not\rightsquigarrow x$. We construct an assignment A that satisfies Φ , as follows.

Repeat the following steps until all vertices have an assignment:

- (a) Pick a vertex (i.e. literal) u such that (i) $A(u)$ has not been defined, and (ii) $u \not\rightsquigarrow \neg u$.³ Let $A(u) = 1$ and $A(\neg u) = 0$.
- (b) For each vertex (i.e. literal) v such that $u \rightsquigarrow v$, let $A(v) = 1$ and $A(\neg v) = 0$. Thus by the symmetry of G_Φ , every vertex reachable from u is assigned 1, and every vertex from which $\neg u$ can be reached is assigned 0.

It is clear that by the hypothesis, the procedure terminates and assigns a value to each literal.

We first show that the assignment A is well defined, that is, for each literal, $A(x)$ cannot be both 0 and 1. It suffices to show that if $u \not\rightsquigarrow \neg u$, then for every $v \in V(G_\Phi)$, $u \not\rightsquigarrow v$ or $u \not\rightsquigarrow \neg v$. Suppose that both $u \rightsquigarrow v$ and $u \rightsquigarrow \neg v$. Then by the symmetry of G_Φ , that $u \rightsquigarrow \neg v$ implies that

²Also note that this condition can be tested in linear time (and space) as the strongly connected components of G_Φ can be computed in linear time (and space), thus 2-SAT can be decided in linear time (and space).

³By the hypothesis, such a vertex (i.e. literal) exists as long as there is an unassigned vertex (i.e. literal).

$v \rightsquigarrow \neg u$. Together with $u \rightsquigarrow v$, this implies that $u \rightsquigarrow \neg u$, contradicting the hypothesis.

We now show that the above assignment A satisfies every clause of Φ , that is, no edge of G_Φ goes from 1 to 0 under assignment A . To show this, it suffices to show that at the beginning of each iteration of the assignment procedure, each vertex reachable from the chosen vertex u either has not been assigned, or has been assigned 1. Suppose that $u \rightsquigarrow v$ and $A(v) = 0$ were already assigned in an earlier iteration. By the construction of the procedure, all vertices from which v can be reached, in particular u , would also have been assigned 0 in the earlier iteration, a contradiction. Hence assignment A satisfies Φ . \square

We now show that 2-SAT is **NL-Hard** by a reduction from $\overline{\text{STCONN}}$ (recall that **NL** = **coNL**).

Let $\langle G, s, t \rangle$ be given, where G is a directed graph, and $s, t \in V(G)$.

- (a) For each edge $(u, v) \in E(G)$, introduce a clause $(\neg u \vee v)$.
- (b) Introduce clauses (s) and $(\neg t)$.

Let Φ be the resulting 2-CNF formula. Clearly Φ can be constructed in log space. We claim that Φ is satisfiable if and only if $s \not\rightsquigarrow t$ in G .

(\Rightarrow): Suppose that $s \rightsquigarrow t$ in G , and consider any assignment to the variables of Φ . In order to satisfy the clauses (s) and $(\neg t)$, variables s and t must be assigned 1 and 0 respectively. Let $s \rightarrow v_1 \rightarrow \dots \rightarrow v_k \rightarrow t$ be a path from s to t . Then in order to satisfy the clauses $(\neg s \vee v_1), (\neg v_1 \vee v_2), \dots, (\neg v_{k-1} \vee v_k)$, variables v_1, \dots, v_k must be assigned 1. But then the clause $(\neg v_k \vee t)$ would not be satisfied.

(\Leftarrow): Suppose that $s \not\rightsquigarrow t$ in G . Define $S = \{u \in V(G) : s \rightsquigarrow u\}$ and $T = \{v \in V(G) : s \not\rightsquigarrow v\}$. Therefore $S \cap T = \emptyset$, $s \in S$, $t \in T$, and for every edge $(u, v) \in E(G)$, exactly one of the following holds: (i) $u, v \in S$, or (ii) $u, v \in T$, or (iii) $u \in T$ and $v \in S$. Let A be the assignment to the variables of Φ (i.e. the vertices of G) that assigns 1 to all vertices in S , and assigns 0 to all vertices in T . Assignment A satisfies Φ , since $A(s) = 1$, $A(t) = 0$, and under assignment A there is no edge in G going from 1 to 0, as there is no edge from T to S . \square

3. Let BIPARTITE be the language consisting of bipartite graphs. Prove that BIPARTITE \in **NL**.

Proof: We show that $\text{BIPARTITE} \in \text{coNL} = \text{NL}$. Recall that a graph is bipartite if and only if it has no cycle of odd length. The following is an NL -Turing machine that decides NON-BIPARTITE :

On input $\langle G \rangle$:

- (a) Nondeterministically choose a start vertex $s \in V(G)$.
- (b) Set the current vertex $u \leftarrow s$.
- (c) Keep a counter k and initially set $k \leftarrow 0$.
- (d) While $k \leq n$:
 - i. Nondeterministically choose a neighbor v of u , set $u \leftarrow v$, increment $k \leftarrow k + 1$.
 - ii. If $v = s$ and k is odd, then output 1 and halt.
- (e) Halt and output 0.

It is clearly that each branch of the TM halts and uses log space. If G has a cycle of odd length, then clearly some branch of the TM accepts. It can be shown that conversely, if some branch of the TM accepts, that is, if there is a vertex s such that there is a walk of odd length that starts from s and returns to s , then G has a cycle of odd length. We leave out the details. \square

4. Let USTCONN be the following problem: Given an *undirected* graph $G = (V, E)$ and two vertices $s, t \in V$, decide whether there is a path from s to t in G . Prove that USTCONN and BIPARTITE are computationally equivalent under log-space reductions.

Proof: The proof given follows the guideline for Exercise 5.16 in Goldreich's textbook. We show that BIPARTITE is log-space reducible to USTCONN , and vice versa.

Let f be the mapping of a graph G to a graph G' , where each vertex $v \in V(G)$ yields two vertices $v^{(1)}, v^{(2)} \in V(G')$, and each edge $\{u, v\} \in E(G)$ yields two edges $\{u^{(1)}, v^{(2)}\}, \{u^{(2)}, v^{(1)}\} \in E(G')$. Clearly, given G , $G' = f(G)$ can be computed in log space (in fact, in constant space).

It can also be seen that G' is bipartite: If G' had a cycle of odd length, then by the construction of G' , such a cycle must be of the form $v_1^{(1)} \rightarrow v_2^{(2)} \rightarrow v_3^{(1)} \rightarrow v_4^{(2)} \rightarrow \dots \rightarrow v_k^{(1)} \rightarrow v_1^{(1)}$, or of the form $v_1^{(2)} \rightarrow v_2^{(1)} \rightarrow v_3^{(2)} \rightarrow v_4^{(1)} \rightarrow \dots \rightarrow v_k^{(2)} \rightarrow v_1^{(2)}$, where k is odd. However G' has no edge of the form $\{v_k^{(1)}, v_1^{(1)}\}$ or $\{v_k^{(2)}, v_1^{(2)}\}$.

Claim 5 *Vertex $v \in V(G)$ lies on a cycle of odd length in G if and only if $v^{(1)}$ and $v^{(2)}$ are connected in G' .*

Claim 6 *Let $s, t \in V(G)$. Then*

- *s and t are connected by a path of even length in G if and only if G' ceases to be bipartite when augmented with the edge $\{s^{(1)}, t^{(1)}\}$; and*
- *s and t are connected by a path of odd length in G if and only if G' ceases to be bipartite when augmented with a new vertex x and edges $\{s^{(1)}, x\}$ and $\{x, t^{(1)}\}$.*

Let \mathcal{O}_{UST} be a membership oracle for **USTCONN**. By Claim 5, the following oracle machine M is a log-space reduction from **BIPARTITE** to **USTCONN**.

$M^{\mathcal{O}_{\text{UST}}}(G)$:

- (a) Construct $G' = f(G)$.
- (b) For each $v \in V(G)$:
 - i. Submit $\langle G', v^{(1)}, v^{(2)} \rangle$ to \mathcal{O}_{UST} , that is, ask \mathcal{O}_{UST} whether $v^{(1)}$ and $v^{(2)}$ are connected in G' .
 - ii. If the answer is YES, then output 0 and halt.
- (c) If the answer from \mathcal{O}_{UST} is NO for each $v \in V(G)$, then output 1.

Let \mathcal{O}_{BIP} be a membership oracle for **BIPARTITE**. By Claim 6, the following oracle machine N is a log-space reduction from **USTCONN** to **BIPARTITE**.

$N^{\mathcal{O}_{\text{BIP}}}(G, s, t)$:

- (a) Construct $G' = f(G)$.
- (b) Obtain $\overline{G'}$ by augmenting G' with the edge $\{s^{(1)}, t^{(1)}\}$.
Ask \mathcal{O}_{BIP} whether $\overline{G'}$ is bipartite.
If the answer is NO, then output 1 and halt.
- (c) Obtain $\widehat{G'}$ by augmenting G' with a new vertex x and edges $\{s^{(1)}, x\}$ and $\{x, t^{(1)}\}$.
Ask \mathcal{O}_{BIP} whether $\widehat{G'}$ is bipartite.
If the answer is NO, then output 1 and halt.

(d) Output 0.

It remains to prove Claims 5 and 6.

Proof of Claim 5:

(\Rightarrow): Suppose that v lies on a cycle $v = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow v_1$ in G , where k is odd. Then by the construction of G' , the path $v_1^{(1)} \rightarrow v_2^{(2)} \rightarrow v_3^{(1)} \rightarrow v_4^{(2)} \rightarrow \cdots \rightarrow v_k^{(1)} \rightarrow v_1^{(2)}$ exists in G' , and thus $v^{(1)}$ and $v^{(2)}$ are connected in G' .

(\Leftarrow): Suppose that there is a path from $v^{(1)}$ to $v^{(2)}$ in G' . By the construction of G' , the path must be of the form $v^{(1)} = v_1^{(1)} \rightarrow v_2^{(2)} \rightarrow v_3^{(1)} \rightarrow v_4^{(2)} \rightarrow \cdots \rightarrow v_k^{(1)} \rightarrow v_1^{(2)} = v^{(2)}$, where k is odd, and $(v = v_1, v_2, \dots, v_k)$ forms a k -cycle in G . \square

Proof of Claim 6:

The proofs of the two parts are similar. We give a proof of the first part.

(\Rightarrow): Let $s = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k = t$ be a path of length k connecting s and t in G , where k is even. By the construction of G' , the path $s^{(1)} = v_0^{(1)} \rightarrow v_1^{(2)} \rightarrow v_2^{(1)} \rightarrow v_3^{(2)} \rightarrow \cdots \rightarrow v_k^{(1)} = t^{(1)}$ of length k exists in G' . Thus adding the edge $\{s^{(1)}, t^{(1)}\}$ results in a cycle of length $k + 1$, and causes $\overline{G'}$ to be non-bipartite.

(\Leftarrow): Suppose that adding the edge $\{s^{(1)}, t^{(1)}\}$ in G' cause $\overline{G'}$ to be non-bipartite. This happens if and only if the $\{s^{(1)}, t^{(1)}\}$ introduces a cycle of odd length. By the construction of G' , such a cycle must be of the form $s^{(1)} = v_0^{(1)} \rightarrow v_1^{(2)} \rightarrow v_2^{(1)} \rightarrow v_3^{(2)} \rightarrow \cdots \rightarrow v_k^{(1)} = t^{(1)} \rightarrow s^{(1)}$, where k is even. Thus the path $s^{(1)} = v_0^{(1)} \rightarrow v_1^{(2)} \rightarrow v_2^{(1)} \rightarrow v_3^{(2)} \rightarrow \cdots \rightarrow v_k^{(1)} = t^{(1)}$ exists in G' . By the construction of G' , the path $s = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k = t$ of length k exists in G . \square

Problem 8

1. Let $s(n)$ be a space-constructible function. Prove that a language $L \in \mathbf{NSPACE}(s(n))$ if and only if there is a deterministic Turing machine M with a read-once⁴ certificate tape such that for every $x \in \{0, 1\}^*$, $x \in L$ if and only if there is a certificate $y \in \{0, 1\}^{2^{O(s(|x|))}}$ such that $M(x, y) = 1$, where x is placed on M 's input tape, y is placed on M 's read-once certificate tape, and M uses $O(s(|x|))$ space.

Proof:

(\Rightarrow): Let $L \in \mathbf{NSPACE}(s(n))$, and let N be a nondeterministic Turing machine that decides L in $O(s(n))$ space. For each $x \in L$, let its certificate π_x be the sequence of configurations (C_0, \dots, C_m) of an accepting branch of N on x , where $m \leq 2^{O(s(n))}$, and $|C_i| = O(s(n))$ for each i .

The following Turing machine M decides L in $O(s(n))$ space, with a read-once certificate of length at most $2^{O(s(n))}$:

- (a) Interpret the given alleged certificate as a sequence of configurations (C_0, \dots, C_m) of a branch of N on x .
- (b) Read C_0 from the certificate and copy it to the work tape.
- (c) Check whether C_0 is the initial configuration of N on x . If not, reject and halt.
- (d) For each $1 \leq i \leq m$:
 - i. Read C_i from the certificate and copy it to the work tape.⁵
 - ii. Check whether C_{i-1} legally yields C_i by the transition function of N . If not, then reject and halt.
- (e) Check whether C_m is an accepting configuration. If so, accept; otherwise reject.

Clearly, $x \in L$ if and only if there is a certificate π_x of length at most $2^{O(s(n))}$ such that $M(x, \pi_x) = 1$. It is also clear that M reads the certificate once. Since only two configurations need to be stored on the work tape at any time, M uses $O(s(n))$ space.

⁴That is, the head of the tape moves only from left to right.

⁵Note that space shall be reused — if $i > 1$, then C_i shall replace C_{i-2} on the work tape.

(\Leftarrow): Suppose that for L there is a Turing machine M as described. The following nondeterministic Turing machine decides L : Simply simulate M , and whenever M needs to access the certificate, nondeterministically guess the current bit of the certificate and give it to M . Clearly N has a branch that accepts x if and only if $x \in L$. Since the certificate is $2^{O(s(n))}$ bits long, M needs $O(s(n))$ bits to simulate the position of the head to the certificate tape, and since the certificate is *read-once*, M just needs $O(1)$ space to simulate the content of the certificate. \square

- Does the above result hold if the head of the certificate tape of M is allowed to move in both directions? Justify your answer.

Answer: The above is unlikely to hold, since if the certificate is not read-once, then the class of languages with $O(s(n))$ -space verifiers is exactly $\mathbf{NTIME}(2^{O(s(n))})$. This is unlikely the class $\mathbf{NSPACE}(s(n))$ which is contained in $\mathbf{DTIME}(2^{O(s(n))})$.

We show this for the special case where $s(n) = \log n$. Let C be the class of log-space verifiable languages where the certificate is *not* read-once. Since $\mathbf{L} \subseteq \mathbf{P}$, it follows that $C \subseteq \mathbf{NP}$.

We now show that $\mathbf{NP} \subseteq C$. First it is not difficult to see that the Cook-Levin reduction, from any \mathbf{NP} language to 3-SAT, can be computed in *log-space*. Therefore, every language in \mathbf{NP} is log-space reducible to 3-SAT. We next observe 3-SAT is log-space verifiable: For a satisfiable 3-CNF formula Φ , the certificate is just a satisfying assignment, and for each clause, the verifier simply reads the assignments to the variables in the clause and checks whether the clause is satisfied by the assignment. Since log space is preserved under composition, it follows that every language in \mathbf{NP} is contained in C . \square

Problem 9

Prove that $\mathbf{P} \neq \mathbf{SPACE}(n)$.

Proof: The trick is to apply *padding*. For each string x and natural number ℓ , define

$$\text{pad}(x, \ell) \stackrel{\text{def}}{=} x \circ \#^\ell,$$

where the notation \circ denotes concatenation, $\#$ is a special symbol, and $j = \max\{\ell - |x|, 0\}$. For a language A and a function $f : \mathbb{N} \rightarrow \mathbb{N}$, define

$$\text{pad}(A, f(n)) \stackrel{\text{def}}{=} \{\text{pad}(x, f(|x|)) : x \in A\}.$$

Claim 7 *Let A be a language. If $\text{pad}(A, n^k) \in \mathbf{P}$ for some constant k , then $A \in \mathbf{P}$.*

Proof: If $\text{pad}(A, n^k)$ can be decided in time $O(n^t)$, then A can be decided in time $O((n^k)^t) = O(n^{kt})$. \square

Assume for the sake of contradiction that $\mathbf{P} = \text{SPACE}(n)$. By the Space Hierarchy Theorem, there is a language B such that $B \in \text{SPACE}(n^2)$ but $B \notin \text{SPACE}(n)$. Since $B \in \text{SPACE}(n^2)$, it follows that $\text{pad}(B, n^2) \in \text{SPACE}(n)$, and thus by the hypothesis $\text{pad}(B, n^2) \in \mathbf{P}$. This implies, by Claim 7, that $B \in \mathbf{P}$, and hence again by the hypothesis, $B \in \text{SPACE}(n)$, a contradiction. Therefore, $\mathbf{P} \neq \text{SPACE}(n)$. \square

Remark: We only know that $\mathbf{P} \neq \text{SPACE}(n)$, but do *not* know whether either side is more powerful than the other.

Problem 10

Prove that there exist oracles A and B such that $\mathbf{NP}^A = \mathbf{coNP}^A$ and $\mathbf{NP}^B \neq \mathbf{coNP}^B$.

Proof: It is clear that $\mathbf{NP}^{\text{TQBF}} = \mathbf{coNP}^{\text{TQBF}} = \mathbf{PSPACE}$. We prove next the existence of an oracle B such that $\mathbf{NP}^B \neq \mathbf{coNP}^B$. The proof is similar to that for the oracle separation of \mathbf{P} and \mathbf{NP} .

For an oracle A , define the language

$$L_A \stackrel{\text{def}}{=} \{1^n : \forall x \in A, |x| \neq n\}.$$

It is clear that for every oracle A , $L_A \in \mathbf{coNP}^A$: On input 1^n , an NTM simply guesses a string $x \in \{0, 1\}^n$, asks whether $x \in A$, and accepts if and only if $x \notin A$. Clearly, if $x \in L_A$, then A contains no string of length n and thus all branches accept; if $x \notin L_A$, then A contains some string of length n and thus at least one branch rejects.

We now construct an oracle B such that $L_B \notin \mathbf{NP}^B$. Let M_1, M_2, \dots be an enumerate of all NTMs, in which every NTM appears infinitely many times. Initially letting $B = \emptyset$, we construct B in stages, where in Stage i for each $i \in \mathbb{N}$:

1. Choose $n \in \mathbb{N}$ that is greater than the lengths of all the strings considered in the previous stages, and large enough so that $2^n > in^i$.
2. Simulate (every branch of) $M_i^B(1^n)$ for at most in^i steps. If M_i queries a string y whose status has been determined, then respond consistently. Otherwise, answer NO and declare $y \notin B$.
3. If all branches of M_i reject within in^i steps, then declare $y \notin B$ for each remaining, undeclared string $y \in \{0, 1\}^n$.
4. If all branches of M_i halt within in^i steps and some branch accepts, then declare $y \in B$ for each string $y \in \{0, 1\}^n$ that has not been declared by this accepting branch (even if these strings were declared to be non-members of B by some other branches of M_i).⁶

We show that the language L_B cannot be decided by any nondeterministic polynomial-time oracle machine with access to B . Consider any oracle NTM M that for every n , runs in at most cn^d steps for every string of length n , where c, d are constants. Since M appears infinitely many times in the enumeration M_1, M_2, \dots , there exists $i \in \mathbb{N}$ such that $i \geq c, d$ and $M_i = M$. Let n be the string length chosen in Stage i . Since $cn^d \leq in^i$, the above procedure for constructing B finishes simulating M_i^B in Stage i .

If (every branch of) $M_i^B(1^n)$ rejects, then by construction B contains no string of length n , thus $1^n \in L_B$ and M_i is incorrect.

If some branch of $M_i^B(1^n)$ accepts, since every branch halts in in^i steps and $in^i < 2^n$, there is at least one string $y \in \{0, 1\}^n$ that was not queried by this accepting branch. Therefore by construction B contains at least one string of length n , hence $1^n \notin L_B$ and M_i is incorrect. \square

Problem 11

Prove that for every $k \in \mathbb{N}$, there is a language in **PH** with circuit complexity $\Omega(n^k)$. For extra credit, prove that for every $k \in \mathbb{N}$, there is a language in Σ_2^P with circuit complexity $\Omega(n^k)$.

Note: This result was due to Ravi Kannan.

Proof: Consider any constant $k \in \mathbb{N}$. Let $m = \lfloor (k+1) \log_2 n \rfloor$. By Shannon's theorem, there exists a Boolean function f on m variables that has

⁶Note that this may change the output of other branches of M_i , but does not change the accepting output of the NTM M_i since at least one branch of M_i is accepting, and clearly does not change the output of any M_j with $j < i$.

circuit complexity at least

$$\frac{2^m}{4m} > \frac{n^{k+1}}{8(k+1)\log_2 n} > n^k$$

for sufficiently large n .

For each $m \in \mathbb{N}$, arrange lexicographically the truth tables of all Boolean functions on m variables, and let f_n be the first function in this order that has circuit complexity greater than n^k . We also view f_n as a function on n variables that is completely determined by the first m variables. Thus the language $L \stackrel{\text{def}}{=} \{x : f_{|x|}(x) = 1\}$ has circuit complexity $\Omega(n^k)$.

Claim 8 *The language $L \in \Sigma_3^P$.*

This is because for $x \in \{0, 1\}^n$, $x \in L$ if and only if

- \exists a (truth table for the) Boolean function f_n^7 s.t. $f_n(x) = 1$, and
- \forall circuit C of size at most n^k , $C(y) \neq f_n(y)$ for some $y \in \{0, 1\}^m$,⁸ and
- \forall Boolean function g on m variables s.t. $g < f$ lexicographically,
- \exists a circuit C' s.t. C' is of size at most n^k , and $C'(y) = g(y)$ for every $y \in \{0, 1\}^m$.⁹

The first quantifier is \exists , and the number of alternating quantifiers is 3. \square

⁷Since f_n is determined by the first m variables and $m = \lfloor (k+1)\log_2 n \rfloor$, the truth table of f_n has size $2^m = O(n^{k+1})$.

⁸This condition can be checked in time $O(n^{2k+1})$.

⁹This condition can be checked in time $O(n^{2k+1})$.