Show that the following problems are NP-complete

April 7, 2018

Below is a list of 30 exercises in which you are asked to prove that some problem is NP-complete. The goal is to better understand the theory and to train to recognize to construct reductions. Except for some problems in the first section, the idea is that you reduce each problem either from the problem of the title of the section, or from one of the previous problems in the section.[1]

In the lectures we have shown that the problem 3SAT is NP-complete. This is the famous Levin-Cook theorem. In the first 2 subsections, we prove this again using a large number of reductions between closely related problems. Solutions are given in the end, but note that you only learn to solve exercises if you first try to solve them yourself.

Unless stated otherwise, the input is represented as follows: numbers are given in binary, graphs are given as an incidence matrix, and sets are always finite and given by an explicit list of all the elements. Several inputs are represented by concatenating the representations with a special separation symbol in between. By default graphs are undirected (as usual).

The unary representation of a nonnegative integer n is given by the string 1^n, i.e., the string of length n containing only 1s.

Prove NP-completeness directly from the definition

The following exercises are essentially the first part of the Levin-Cook theorem.

Exercise 1. The problem ACCEPTS SOMETHING FAST:

Input:	A deterministic Turing machine M and an integer t given in unary.
Question:	Does M accept some input after less than t computation steps?

Note that it suffices to consider inputs of length at most t, because in $t-1$ computation steps a machine can read at most the first t bits of the input.

For the definition and examples of circuits we refer to Sipser's book p380.

Exercise 2. The problem CIRCUIT SAT:

Input:	A circuit ϕ.
Question:	Is ϕ satisfiable, i.e., are there Boolean values $(x_1, \ldots, x_n)$ such that $\phi(x_1, \ldots, x_n) = 1$?

Reductions from circuit SAT

Exercise 3. The problem DOUBLE CIRCUIT SAT:

Input:	A circuit ϕ.
Question:	Does ϕ have at least two satisfying assignments?

Exercise 4. The problem 3CONSTRAINT SATISFIABILITY or 3CSAT:

[1] The appendix of the book Michael, R. Garey, and S. Johnson David. "Computers and intractability: a guide to the theory of NP-completeness." WH Free. Co., San Fr (1979).

Input: A Boolean function $f(x_1, \ldots, x_m) = \bigwedge_{i=1}^{\ell} g_i(y_{i,1}, y_{i,2}, y_{i,3})$
where $g_i\colon \{0,1\}^3 \to \{0,1\}$ and $y_{i,j} \in \{x_1, \ldots, x_m\}$.
Question: Is f satisfiable?

Exercise 5. The problem 3SAT:
Input: A formula ϕ in 3CNF.
Question: Is ϕ satisfiable?

The 2SAT problem, i.e., the analogue problem with formulas in 2CNF, can be solved in linear time (this can be easily seen if you know about resolution: a resolution step generates clauses of length 2 from clauses of length 2). The 3SAT problem is much harder. We now finish the proof of the famous Levin-Cook problem: 3SAT is NP-complete. This implies that if the 3SAT problem can be solved in polynomial time, then $\mathsf{P} = \mathsf{NP}$.

Reductions from 3SAT

The following 2 exercises are useful in many applications to prove that some problem is NP-hard. We have discussed them in detail during the class and the solution is given at the end of the notes.

Exercise 6. A *not-all-equal* assignment of a formula in CNF (or 3CNF) is an assignment for which every clause contains at least 1 true literal and at least 1 false literal.

The problem NOT-ALL-EQUAL 3SAT:
Input: A formula ϕ in 3CNF.
Question: Does ϕ have a not-all-equal assignment?

Hint: Let y be a new variable. First reduce 3SAT to NOT-ALL-EQUAL 4SAT by adding y to every clause.

Exercise 7. The problem 1-IN-3 SAT:
Input: A formula ϕ in 3CNF.
Question: Does ϕ have a satisfying assignment so that every clause contains exactly one true literal.

Hint: Reduce from 3SAT. Create for every clause 8 new variables $y_{000}, y_{001}, \ldots, y_{111}$ and reduce 3SAT to 1-IN-AT-MOST-7 SAT. The idea is that precisely 1 of these new variables should be true for each assignment of the clause.

Exercise 8. The problem 0-1 INTEGER PROGRAMMING:
Input: A matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$.
Question: Is there an $x \in \{0,1\}$ such that $Ax \leq b$.

Hint: Reduce from 3SAT.

Exercise 9. The problem BINARY QUADRATIC PROGRAMMING:
Input: A matrix $A \in \{0,1\}^{n \times n \times n}$ and a vector $b \in \{0,1\}^n$.
Question: Is there an $x \in \{0,1\}^n$ such that $\sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j,k} x_i x_j = b_k \bmod 2$.

Hint: Reduce from 3SAT and use that $u \vee v = u + v - uv$ for $u, v \in \{0,1\}$ and $u^2 = u$.

Exercise 10. (This is Ex. 7.28 in Sipser's book.) In the following solitaire game, you are given an $m \times m$ board. On each of its m^2 positions lies either a blue stone, a red stone, or nothing at all. You play by removing stones from the board. You win if each column contains only stones of a single color and each row contains at least one stone.

The problem WINNABLE SOLITAIRE GAME:
Input: An initial configuration of the solitaire game.
Question: Is it possible to win the game?

Exercise 11. The problem AUTOMATON REJECTS SOME SMALL STRING:

Input:	An NFA M and an integer n in unary.
Question:	Is there a string of length n that is rejected by M.

Note that it is not trivial that this problem is in NP. Hint: Reduce from 3SAT. For each formula $\phi(x_1, \ldots, x_n)$ build an automaton that has an assignment $w_1 w_2 \ldots w_n$ as input and accepts if $\phi(w_1, \ldots, w_n)$ is false.

Later we show that if we drop the bound n on the length of the rejected string, the problem becomes PSPACE-complete.

Exercise 12. The problem DIFFERENT FINITE REGULAR LANGUAGES:

Input:	NFAs K and M.
Question:	Do K and M recognize finite languages and are these languages different?

Hint: The hardest part of the exercise is to show that this problem is in NP. First observe that automata recognizing finite languages are essentially cycle free. For the reduction, use that the problem in exercise 11 remains NP-hard if we additionally require that M rejects all strings of length different from n.

Reductions from 1-in-3 SAT

Exercise 13. A *partition* of a set U is a collection of sets $S_1, \ldots, S_e \subseteq U$ that are pairwise disjoint such that $\bigcup_{i=1}^{e} S_i = U$.

The problem CONTAINS PARTITION:

Input:	A set U and a collection C of subsets of U.
Question:	Is there a subcollection $C' \subseteq C$ that is a partition of U?

Exercise 14. The problem SUBSET SUM:

Input:	$X \subseteq \mathbb{Z}$ and $k \in \mathbb{Z}$.
Question:	Does there exist a subset $I \subseteq X$ such that $\sum_{i \in I} i = k$.

Hint: reduce from CONTAINS PARTITION by considering expansions of natural numbers in some base.

Exercise 15. The problem SET PARTITION:

Input:	Positive integers $x_1, \ldots, x_n$.
Question:	Is there a set $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} x_i = \sum_{i \notin I} x_i$?

Exercise 16. The problem DISJOINT SETS:

Input:	A collection C of sets and a number k.
Question:	Are there k sets in C that are pairwise disjoint?

Hint: reduce from 1-IN-3 SAT in a similar way as the problem CONTAINS PARTITION.

Reductions from clique

In class we proved that 3SAT reduces to the following problem. See also Theorem 7.32 p302 in Sipser's book.

The problem CLIQUE:

Input:	A graph (V, E) and an integer k.
Question:	Are there k vertices in V that are pairwise connected?

Exercise 17. The problem INDEPENDENT SET:

Input:	A graph $G = (V, E)$ and an integer k.
Question:	Is there a subset $U \subseteq V$ of k vertices that are pairwise unconnected, i.e., $(U \times U) \cap E$ is empty?

Exercise 18. *A vertex cover* of a graph G is a subset S of vertices of G such that each edge of G has at least one endpoint in S.

The problem VERTEX COVER:

Input:	A graph G and an integer k.
Question:	Does G have a vertex cover of size at most k?

Exercise 19. The problem SET COVER:

Input:	A set E, a list $S_1, \ldots, S_m$ of subsets of E and an integer k.
Question:	Are there k sets in the list whose union equals E?

Exercise 20. The problem SET PACKING:

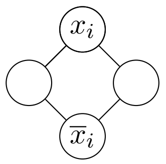
Input:	A set E, a list $S_1, \ldots, S_m$ of subsets of E and an integer k.
Question:	Are there k sets in the list that are pairwise disjoint?

Reductions from 3SAT: graph problems and gadgets

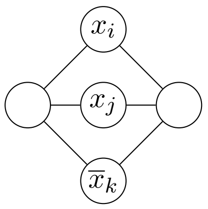
Exercise 21. The problem PATH WITH FORBIDDEN PAIRS:

Input:	A graph $G = (V, E)$, vertices $s, t \in V$, and a set of pairs $(u_1, v_1), \ldots, (u_e, v_e) \in V \times V$.
Question:	Does there exist a path from s to t that visits at most one vertex of each pair (u_i, v_i)?

Hint: see the gadgets below.



variable



clause $x_i \wedge x_j \wedge \overline{x}_k$

The following 2 problems were discussed in detail in class. You can find the reductions in Sipser's book.

Exercise 22. The problem DIRECTED HAMILTONIAN PATH:

Input:	A directed graph $G = (V, E)$, and vertices $s, t \in V$.
Question:	Is there a path from s to t that visits all nodes in V exactly once.

Exercise 23. The problem HAMILTONIAN PATH:

Input:	An (undirected) graph $G = (V, E)$, and vertices $s, t \in V$.
Question:	Is there a path from s to t that visits all nodes in V exactly once.

Exercise 24. A *cycle* in a graph is a path for which the start and end nodes are equal.
The problem DIRECTED HAMILTONIAN CYCLE:

Input:	A directed graph $G = (V, E)$.
Question:	Is cycle that visits all nodes in V exactly once.

Exercise 25. The problem HAMILTONIAN CYCLE:

Input:	An (undirected) graph $G = (V, E)$.
Question:	Is there a path from s to t that visits all nodes in V exactly once.

This problem is perhaps the most cited NP-complete problem in the popularizing literature.

Exercise 26. The problem TRAVELLING SALES MAN:

Input:	A list of all pairwise distances between n points and an integer k.
Question:	Is their a cycle of length at most k that contains all points? In other words, is there a list $p_1, p_2, \ldots, p_m$ containing all points at least once such that $d(p_1, p_2) + \cdots + d(p_{n-1}, p_n) + d(p_m, p_1) \leq k$.

For simplicity of the exercise, we do not require that the distances satisfy the conditions of a metric.

Reductions from not-all-equal 3SAT using gadgets

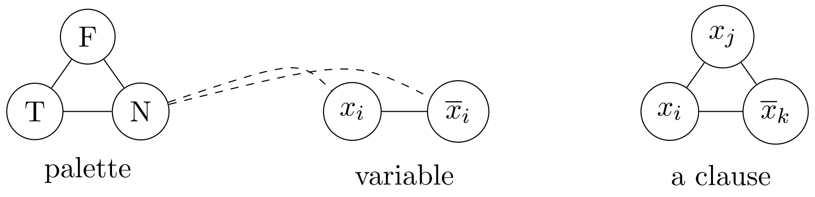
The following problem is important to prove that finding optimal weights in a neural network is NP-hard. (Unfortunately, we did not have time to make the connection with neural networks.)

Exercise 27. A *coloring* of a graph is an assignment of colors to nodes such that no two connected nodes have the same color.

The problem 3COLORABLE:

Input:	A graph G.
Question:	Is G colorable with 3 colors?

Hint: Let the 3 colors be T, F and N. Interpret the first as `True` and `False`, and represent them on a palette.



Reductions from 3Colorability

Exercise 28. The problem CLIQUE COVER:

Input:	A graph $G = (V, E)$ and an integer k.
Question:	Can we partition the vertices in k sets such that each part forms a clique in G?

Hint: choose $k = 3$.

Exercise 29. Consider the following scheduling problem. You are given a list of final exams $F_1, \ldots, F_k$ to be scheduled, and a list of students $S_1, \ldots, S_\ell$. Each student is taking some specified subset of these exams. You must schedule each exam into exactly one slot such that no student is required to take two exams in the same slot. The problem is to determine if such a schedule exists that uses only h slots. Formulate this problem as a language and show that it is NP-complete. Ex. 7.30.

Some other NP-complete and NP-hard problems

Finally we mention some interesting problems for which the reductions are too hard to cover in the lectures.

The problem QUADRATIC DIOPHANTINE EQUATIONS:

Input:	$a, b, c \in \mathbb{Z}_{>0}$.
Question:	Does there exist $x, y \in \mathbb{Z}_{>0}$ such that $ax^2 + by = c$?

The following problem is a simplified version of an important problem in biochemistry. Completeness for the simplified model is often used to argue that the original problem is very difficult.

The problem PROTEIN FOLDING:

Input:	A graph $G = (V, E)$.
Question:	Is it possible to place all vertices of V on different points of a rectangular grid such that adjacent nodes in V are adjacent in the grid.

In the next problem we consider neural networks where every node computes a linear threshold function with rational coefficients. Moreover, our neural networks have only 1 hidden layer that contains 3 nodes. For $w \in \mathbb{Q}^n$, let

$$f_w(x) = \begin{cases} 1 & \text{if } w_1 x_1 + \cdots + w_n x_n + w_{n+1} \\ 0 & \text{otherwise} \end{cases}.$$

The problem PERFECT CLASSIFICATION WITH 3-NODE NEURAL NETS:

Input:	Sets S_+ and S_- of elements in $\{0,1\}^n$ for some n.
Question:	Does there exist $u, v, w, r \in \mathbb{Q}^n$ such that the $f_r\left([f_u(x), f_v(x), f_w(x)]\right)$ equals 1 for all $x \in S_+$ and 0 for all $x \in S_-$?

This problem is also NP-complete for any neural network with a constant number of nodes in a single hidden layer. But this result might not really convince us that optimizing weights of larger neural networks is hard in practice. Usually, people use a large number of internal nodes and use multiple layers. Maybe finding optimal weights in such networks is easier? If the number of layers is unbounded, then NP-hardness is again easy to prove by reducing from CIRCUIT SAT. But also for networks of small depth there exist intractability results, but they require cryptographic assumptions that are stronger than $\mathsf{P} \neq \mathsf{NP}$.

The following 2 problems are important for machine learning. They are not known to be in NP, but is known that they are NP-hard.

The problem CLUSTERING:

Input:	An integer k, a finite set $X \subseteq \mathbb{Q}^d$, and a rational number R.
Question:	Are there k points $c_1, \ldots, c_k$ such that

$$\sum_{x \in X} \min \left\{ \|x - c_i\|_2^2 : i = 1, \ldots, k \right\} \;\leq\; R\ ?$$

This problem even remains NP-hard if we fix $k = 2$ or if we fix $d = 2$.

The problem NONNEGATIVE MATRIX FACTORIZATION:

Input:	An integer k and a matrix $A \in \mathbb{Q}_{\geq 0}^{n \times m}$.
Question:	Does there exist matrices $H \in \mathbb{Q}_{\geq 0}^{n \times k}$ and $W \in \mathbb{Q}_{\geq 0}^{k \times m}$ such that $A = HW$?

One might wonder why these problems are not in NP: why can we not use the centers or the nonnegative matrices as certificates? The problem is that the these objects consist of rational numbers and it can happen that the nominators and denominators are too large to have representations of polynomial size.

Solutions

Exercise 1. To see that ACCEPTS SOMETHING FAST is in NP we use the certificate definition. The certificates of a pair $(M, t) \in$ ACCEPTS SOMETHING FAST, are given by the strings c of length at most t that are accepted by M in time less than t. On input (M, t) and a certificate c, the verifier simulates M during at most $t - 1$ computation steps and checks whether M accepts its input. This can be done in time polynomial in t and the representation size of M. We conclude that the problem is in NP.

We now prove that every language reduces to this problem. Let L be a language in NP and V be a polynomial time verifier for L. Recall that $V(x, c)$ halts in time polynomial in $|x|$ for all c. The reduction from L to ACCEPTS SOMETHING FAST is given by the mapping

$$x \quad \longrightarrow \quad \Big(M_x,\ q(|x|)\Big)$$

where M_x is a machine that on input c simulates $V(x, c)$, and $q(n)$ is a polynomial that bounds the computation time of M_x for all certificates of $x \in L$ with $|x| = n$. Because $V(x, c)$ runs in time polynomial in $|x|$ we can assume that the polynomial bound q exists.

The mapping above can be computed in time polynomial in $|x|$. It remains to show that it is indeed a reduction. By definition of a verifier

$$x \in L \quad \Longleftrightarrow \quad \exists c\, [V(x, c) \text{ accepts}]\,.$$

By construction of M_x and q

$$\begin{aligned} &\Longleftrightarrow && M_x \text{ accepts some } c \text{ of size at most } q(|x|) \text{ in time } q(|x|). \\ &\Longleftrightarrow && (M_x, q(|x|)) \in \text{ACCEPTS SOMETHING FAST}. \end{aligned}$$

Exercise 2. Clearly this problem is in NP, because the assignments that make ϕ equal to 1 are certificates. To prove completeness we reduce from ACCEPTS SOMETHING FAST.

Recall that a machine state is described by a triple in $\mathbb{Z}_{\geq 0} \times Q \times \Gamma^\infty$, where Q are the control states and Γ is the tape alphabet. Because we are interested in computations that terminate in time less than t, we only bother with the first t cells of the tape and describes machine states as elements from $\mathbb{Z}_{\geq 0} \times Q \times \Gamma^t$. The triple can be naturally represented as a concatenation of 3 binary strings.

It is easy to believe that there exist 3 circuits I, ϕ and H of size polynomial in t and the description length of M, such that

- I maps an input string x to a representation of the start state with input x,
- ϕ maps the representation of a triple (i, q, T) to the representation of the next state (j, r, S) after a computation step,
- H maps every representation of an accept state to 1 and all other inputs to 0.

Consider the circuit $\psi_{M,t}$ given by the concatenation of I with t concatenations of ψ and H. This circuit has size polynomial in t and the representation size of M. Moreover, it maps x to 1 if and only if M accepts x in time at most t. Hence, the mapping $(M, t) \mapsto \psi_{M,t}$ is a polynomial time reduction from ACCEPTS SOMETHING FAST to CIRCUIT SAT.

Exercise 4. We reduce from CIRCUIT SAT. Assume some circuit has m nodes of which n are input nodes. For all nodes that are not an input nodes, we create a new variable. Let these variables be $x_{n+1}, x_{n+2}, \ldots, x_m$. Given some input values $x_1, \ldots, x_n$, all values $x_{n+1}, \ldots, x_m$ are fixed. Every non-input node x_i is computed using the values of at most 2 other nodes, say x_j and x_k, using $\wedge$, $\vee$ or not operations. The condition $x_i = x_j \wedge x_k$ is equivalent to the requirement $g(x_i, x_j, x_k) = 1$ for some function g. Similarly for $x_j \vee x_k$. For a negation we can define a function g after adding a 3th dummy variable (g does not depend on this 3th variable). The satisfiability of all these requirements, together with the requirement that the variable of the output node equals 1, is equivalent to the satisfiability of the original circuit.

Exercise 5. Recall that a formula in 3CNF is a formula of the form

$$\phi(x_1, \ldots, x_m) = \bigwedge_{i=1}^{\ell} \big(y_{i,1} \vee y_{i,2} \vee y_{i,3}\big).$$

with $y_{i,k} \in \{x_1, \overline{x}_1, \ldots, x_m, \overline{x}_m\}$ for $i = 1, \ldots, \ell$ and $k = 1, 2, 3$. The disjunctions $y_{i,1} \vee y_{i,2} \vee y_{i,3}$ are called *clauses* of ϕ. We use the observation that every Boolean function with inputs of length k can be written as 2^k conjunctions of k disjunctions. Afterwards, we apply this for $k = 3$.

Let w be a string of length k, let $\text{not}^0\, x_i = x_i$ and let $\text{not}^1\, x_i = \overline{x}_i$.

$$(\text{not}^{w_1}\, x_1) \vee \cdots \vee (\text{not}^{w_k}\, x_k) \quad = \quad \mathbf{1}_{x \neq w} \quad = \quad \begin{cases} 0 & x = w \\ 1 & \text{otherwise.} \end{cases}$$

Thus any Boolean function h can be written as

$$\begin{aligned} h(x) \;&=\; \min\{\mathbf{1}_{x\neq w} : h(w) = 0\} \\ &=\; \bigwedge_{w\colon h(w)=0} \Big((\text{not}^{w_1}\, x_1) \vee \cdots \vee (\text{not}^{w_k}\, x_k)\Big). \end{aligned}$$

Exercise 6. Let ϕ be a formula in 3CNF. Let $f(\phi)$ be the formula in 4CNF obtained by adding y to every clause. We show that f is a reduction from 3SAT to NOT-ALL-EQUAL 4SAT. Suppose ϕ is satisfiable, then the this assignment together with $y = 0$ is an assignment in which every clause has 2 different literals. Suppose that $f(\phi)$ has an assignment in which every clause has 2 different literals. Note that if each variable is negated, we also obtain such an assignment. Hence, there is such an assignment in which $y = 0$. For this assignment, every clause has a true literal, hence ϕ is satisfiable. Thus 3SAT $\leq_p$ NOT-ALL-EQUAL 4SAT.

Now we reduce NOT-ALL-EQUAL 4SAT to NOT-ALL-EQUAL 3SAT. We map each clause of size 4 to 2 clauses of size 3 using a new variable a:

$$\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4 \quad \longrightarrow \quad \ell_1 \vee \ell_2 \vee a \text{ and } \overline{a} \vee \ell_3 \vee \ell_4.$$

(A formula in 4CNF with m clauses and n variables is mapped to a formula in 3CNF with $2m$ clauses and $n + m$ variables.)

We show that this mapping is a reduction. Given a not-all-equal assignment for the formula in 4CNF, we need to create a not-all-equal assignment for the formula in 3CNF. For this we need to fix the value of the variable a corresponding to each clause $\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4$. This is done as follows. If $\ell_1 = \ell_2$, we choose a to be the opposite of this value. If $\ell_3 = \ell_4$, we choose a equal to this value. Otherwise, the value of a is not important. If $\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4$ has two different values, then both clauses have 2 different values.

Now assume some that for every assignment of the 4CNF formula there is a clause whose values are all equal. Then it is easy to see that that 1 of the 2 corresponding clauses in the 3CNF formula must have all equal values, regardless of the choice of a. Thus NOT-ALL-EQUAL 4SAT $\leq_p$ NOT-ALL-EQUAL 3SAT.

Exercise 7. For every clause $\ell_1 \vee \ell_2 \vee \ell_3$ we create 8 new variables $y_{000}, y_{001}, \ldots, y_{111}$. The idea is that an assignment of the first clause corresponds to an assignment where all these y-variables are 0 except for $y_{\ell_1\ell_2\ell_3}$. This is done as follows. A clause $\ell_1 \vee \ell_2 \vee \ell_3$ is mapped to 7 clauses. The first clause contains all new variables except y_{000}.

$$y_{001} \vee y_{010} \vee y_{100} \vee y_{011} \vee y_{101} \vee y_{110} \vee y_{111}$$

The next 2 clauses use the literal ℓ_1:

$$\ell_1 \vee y_{000} \vee y_{001} \vee y_{010} \vee y_{011}$$
$$\overline{\ell}_1 \vee y_{100} \vee y_{101} \vee y_{110} \vee y_{111}.$$

We add 2 similar clauses both for the literal ℓ_2 and ℓ_3.

If $\ell_1 \vee \ell_2 \vee \ell_3$ is true, then by setting the variable $y_{\ell_1\ell_2\ell_3}$ true and all others false, each of the 7 clauses contains precisely 1 true literal. Oppositely, if each clause contains precisely 1 true literal, then precisely 1 of the variables from the first clause must be true, and by the 6 other clauses, this variable must be consistent with the values of ℓ_1, ℓ_2 and ℓ_3. This implies that $\ell_1 \vee \ell_2 \vee \ell_3$ is true.

Now in a similar way as in the previous exercise, we can make the clauses smaller by adding new variables.

Exercise 8. We map every clause $\ell_1 \vee \ell_2 \vee \ell_3$ in a formula ϕ in 3CNF to a linear equation $\ell_1 + \ell_2 + \ell_3 \geq 1$ by rewriting negative literals $\ell_i = \overline{x}_i$ as $1 - x_i$. In this way, the set of clauses in ϕ is mapped to a system of linear equations. An assignment with $x_i \in \{0, 1\}$ is a solution of the system if and only if every clause contains a true literal.

Exercise 9. Again we map negative literals $\ell_i = \overline{x}_i$ to terms $1 - x_i$. The requirement that a clause $\ell_1 \vee \ell_2 \vee \ell_3$ is true is equivalent to

$$\begin{array}{rcl} \ell_1 \vee \ell_2 & = & a \\ a \vee \ell_3 & = & 1 \end{array} \qquad \Longleftrightarrow \qquad \begin{array}{rcl} \ell_1 + \ell_2 - \ell_1\ell_2 & = & a \\ a + \ell_3 - a\ell_3 & = & 1. \end{array}$$

After expanding (the negations) we obtain a set of quadratic equations and we use the equality $u^2 = u$ to remove all linear terms. By construction an assignment makes ϕ true if and only if it is a solution of the equations.

Exercise 11. The certificates are given by strings of length at most n that are rejected by M. The verifier runs the algorithm discussed in the first lecture notes (also known as the finger algorithm).

To reduce 3SAT to this problem, consider a formula ϕ in 3CNF with n variables and ℓ clauses. Every assignment is represented by an n-bitstring and these strings corresponds to inputs for the automaton. First we build for each clause a machine that accepts all strings in which the clause is false. Each such machine can be constructed using at most $2(n+1)$ states. Then we take the union of all these automata. The union is taken using some nondeterministic splitting in the start node with ε-edges. We obtain an automaton M with at most $2\ell(n+1)$ states. By construction, this automaton accepts every assignment that makes the formula false, but rejects an assignment that makes the formula true.

Exercise 12. Note that if a nondeterministic automaton has a state from which there exists no path to an accept state, then we can remove this state without changing the language recognized by the automaton. Assume that from every state of some automaton there exists a path to an accept state. This automaton recognizes a finite language if and only if it has no cycles.

The certificates (that show that the 2 automata are different) are given by the strings that are rejected by one automaton and accepted by the other, together with a bit that indicates which automata accepts and rejects. The verifier first checks that K and M recognize finite languages using the property described above. Then it checks that the indicated automaton accepts the string. Finally it uses the finger algorithm from our lectures about automata to check that the other automaton rejects the string.

Because the automata in the language are equivelent to some cycle free automata, the maximal length of an accepted string is bounded by the number of states. Hence, the certificated has polynomial size, and all checks executed by the verifier can be done in polynomial time.

To show completeness, we use the previous problem with the additional condition that M rejects all strings whose length differs from n. The reduction in the previous problem can easily be assumed to satisfy this additional condition. Let machine K_n be the machine that accepts precisely the strings of length n. The mapping $(M, n) \mapsto (K_n, M)$ is a reduction from problem 11 to this problem.

Exercise 13. We reduce from 1-IN-3 SAT. Let ϕ be a formula in 3CNF with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_k$. We choose

$$U = \{v_1, v_2, \ldots, v_n, c_1, \ldots, c_\ell\}.$$

We say that the literal ℓ appears in a clause c if $c = \ell_1 \vee \ell_2 \vee \ell_3$ and $\ell \in \{\ell_1, \ell_2, \ell_3\}$. We define $2n$ subsets S_ℓ:

$$S_{x_i} = \{v_i\} \cup \{c_j \colon x_i \text{ appears in } c_j\}$$
$$S_{\overline{x}_i} = \{v_i\} \cup \{c_j \colon \overline{x}_i \text{ appears in } c_j\}.$$

We prove that the mapping $\phi \mapsto (U, \{S_{x_1}, S_{\overline{x}_1}, \ldots, S_{x_n}, S_{\overline{x}_n}\})$ is a polynomial time reduction. Clearly the pair has a polynomial representation size and can be computed in polynomial time. Suppose ϕ has a 1-in-3 assignment $x_1, \ldots, x_n$. Then the collection

$$\mathcal{P}_x = \{S_\ell \colon \text{assignment } x \text{ makes literal } \ell \text{ true}\}$$

is a partition of U. Indeed, for each i:

- Each element v_i appears only in the sets S_{x_i} and $S_{\overline{x}_i}$ and we selected precisely 1 of these sets.
- Each element c_j only appears in 3 sets S_ℓ given by the literals of c_j. Because x is a 1-in-3 assignment, precisely 1 such literal is true and precisely one such set S_ℓ appears in $\mathcal{P}_x$.

Assume the collection contains a partition $\mathcal{P}$. Then for each $i \leq n$, only the sets S_{x_i} and $S_{\overline{x}_i}$ contain v_i and therefore, precisely 1 of them belongs to $\mathcal{P}$. Let $x_i = 1$ if $S_{x_i} \in \mathcal{P}$ and $x_i = 0$ otherwise. In other words, we assign the variables such that any literal ℓ is true if and only if $S_\ell \in \mathcal{P}$. Because every variable c_j appears in precisely one set S_ℓ, every clause contains precisely 1 true variable. In conclusion, x is a 1-in-3 assignment and the mapping above is a reduction.

Exercise 14. We reduce from CONTAINS PARTITION. Let b be the number of sets in C plus 1. (In fact the plus 1 is not needed, but it simplifies the argument.) Assume $U = \{0, 1, \ldots, |U| - 1\}$. To each set $S \in C$ we associate the integer $x_S = \sum\{b^u \colon u \in S\}$. Thus the expansion of x_S in base b contains only 0s and 1s, and 1s only appear precisely at the positions $u \in S$. We choose $k = \sum_{i=0}^{|U|-1} b^i$.

Note that if we add at most $b - 1$ numbers x_S using the expansion in base b, each uth digit equals the sum of the uth digits of the numbers (we never carry anything to the next digit). Hence, the set $\{x_S : S \in C\}$ contains a subset with sum k if and only if C contains a partition of U.

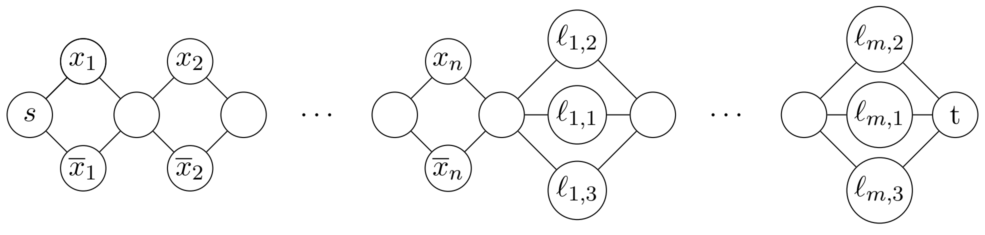
Exercise 17. Consider the inverse graph (V, E), which is given by $(V, (V \times V) \setminus E)$. This graph has an independent set of size k if and only if (V, E) has a clique of size k.

Exercise 18. We reduce from the problem INDEPENDENT SET. Note that if $U \subseteq V$ is an independent set, then $V \setminus U$ is a vertex cover. Hence, (V, E) has an independent set of size at least k if and only if (V, E) has a vertex cover of size at most $|V| - k$.

Exercise 19. We reduce from VERTEX COVER. Consider a graph (V, E) and an integer k. For each $v \in V$ let $S_v \subseteq E$ be the set of edges that are incident on v. Let $C = \{S_v : v \in V\}$. We can cover E with at most k sets S_v if and only if (V, E) has a vertex cover of size at most k.

Exercise 20. We reduce from INDEPENDENT SET. Let (V, E) be a graph. For every $v \in V$, let S_v be the set of edges that are incident on v. U is an independent set if and only if the sets S_u and S_v are disjoint for all $u \in U$ and $v \in U$.

Exercise 21. We concatenate gadgets for all variables followed by gadgets for all clauses. The leftmost node is s and the rightmost node is t. In the following picture this is represented for a formula in 3CNF with n variables and m clauses, where $\ell_{j,k}$ is the kth literal of the jth clause.



The list of forbidden pairs of nodes is given by all pairs (u, v) that are labeled with opposite literals, i.e., by $(x_i, \overline{x}_i)$ or $(\overline{x}_i, x_i)$.

Exercise 24. In the reduction from 3SAT to DIRECTED HAMILTONIAN PATH from Sipser's book, node s only has outgoing edges and t has only incoming edges. We can simply add an edge from t to s to obtain a graph with a Hamiltonian cycle. Note that this edge belongs to every Hamiltonian cycle, because for each node it contains an incoming and outgoing edge.

Exercise 26. We reduce from HAMILTONIAN CYCLE. Consider a graph $G = (V, E)$ and vertices $s, t \in V$. The list of points are given by the vertices V, and the distances are given by

$$d_G(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{otherwise.} \end{cases}$$

The mapping $G \mapsto (d_G, k)$ is a reduction. Indeed, a Hamiltonian cycle contains precisely $|V|$ edges and the length according to d_G is therefore $|V|$. On the other hand, if there is a cycle with d_G-length equal to $|V|$ that visits V nodes, then the d_G-length is the sum of $|V|$ distances. The minimal value of a term in this sum is 1, hence, all distances in the cycle must be one which means that all subsequent pairs are edges from the graph.

Exercise 27. We reduce from NOT-ALL-EQUAL 3SAT. Consider a formula ϕ in 3CNF. We construct a graph G_ϕ that has 3 parts.

- The palette, as indicated in the picture of the hint.
- For each variable we create a variable gadget as indicated in the hint. For a variable x_i the nodes are labeled by x_i and $\overline{x}_i$. Connect both nodes to the N-node of the palette.
- For each clause we create a triangle and label the nodes by the literals of the clause. We connect each node with the node in the variable gadget labeled by the *negated* literal.

Clearly this graph can be build in polynomial time. It remains to show that $\phi \mapsto G_\phi$ is a reduction.

Assume that formula ϕ has a not-all-equal assignment. We need to color G_ϕ. We use colors F, T and N. First color the palette as indicated in the hint. Then color the variable gadgets: nodes labeled by true literals are colored with T and for the false ones we use F. Finally, we color the clause gadgets. In every clause we pick a true and a false literal and color the corresponding nodes with T and F respectively. The 3th node of the gadget is colored with N. Because nodes in variable and clause gadgets are only connected to nodes labeled by opposite literals, this is indeed a coloring.

Now assume graph G_ϕ has a coloring. Let us first rename the colors so that they match the names T, F and N in the palette. Because all nodes of the variable gadgets are connected to N, they are all colored by T and F. To construct the assignment x, we use the variable gadgets. Let x be the assignment for which all nodes labeled by true literals are colored with T and the others colored with F. We show need to show that every clause contains a true and a false literal. Note that in a coloring of a triangle, all colors must appear exactly once. Thus in every clause gadget there must be a node with a color F and a node with color T. Because each literal in the clause gadget is connected to its negated literal in the variable gadget, every literal colored by T must be true. Similarly, every literal colored by F must be false. Hence, every clause contains a true and a false literal.