Harry Langford
hjel2@cam.ac.uk

1 2019 Paper 6 Question 4

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2019p6q4.pdf

A Boolean formula ϕ is in *conjunctive normal form* (CNF) if it is the conjunction of clauses, each of which is the disjunction of literals. It is said to be in k-CNF (for $k \in \mathbb{N}$) if each clause has exactly k literals in it.

An assignment $\sigma : V \to \{\text{true}, \text{false}\}$ of truth values to the variables is a *satisfying assignment* for a CNF formula ϕ if it makes at least one literal in each clause of ϕ true. It is said to be a *not-all-equals* assignment for ϕ if it makes at least one literal in each clause of ϕ true *and* at least one literal in each clauses of ϕ false.

Let CNF-SAT denote the problem of determining, given a formula in CNF, whether it has a satisfying assignment.

Let k-SAT denote the problem of determining, given a formula in k-CNF, whether it has a satisfying assignment.

Let k-NAE denote the problem of determining, given a formula in k-CNF, whether it has a not-all-equals assignment.

(a) Explain why CNF-SAT is NP-complete. Your explanation should include a full definition of NP-completeness and a brief sketch on the proof of the Cook-Levin theorem.

A language L is NP-complete if and only if it is both in NP and is NP-hard. CNF-SAT is both in NP and is NP-hard – therefore it is NP-complete.

A language L is in NP if and only if it is recognisable in polynomial time by a Nondeterministic Turing Machine.

A language L is NP-hard if and only if, for all languages $A \in NP.A \leq_P L$ – all languages in NP are polynomially reducible to L.

A language $L_1 \subseteq \Sigma_1^*$ is polynomially reducible to $L_2 \subseteq \Sigma^*$ if and only if there exists a function $f : \Sigma_1^* \to \Sigma_2^*$ such that $x \in L_1 \iff f(x) \in L_2$.

CNF-SAT is solvable by a Nondeterministic Turing Machine in polynomial time. By nondeterministically writing an interpretation, replacing all variables with their values under the interpretation; and running the (polynomial) algorithm for circuit value problem (specified in the lectures). The Nondeterministic Turing Machine will then accept if and only if the formula is satisfiable.

We can prove that CNF-SAT is NP-hard by reducing the computation of an arbitrary nondeterministic Turing machine into a CNF-SAT problem which is polynomial in the length of the input. This is the approach taken in the standard proof of the Cook-Levin theorem.

Let $H_{t,p}$ hold if the head is in position p at time t. Let $T_{t,p,\sigma}$ be true if at time t the tape cell at position p contains symbol σ. Let $S_{t,q}$ hold if at time t the state of the machine is q.

We can use these definitions to encode formulae which represent the computation of an arbitrary Turing machine. We can then and the expression with conditions which restrain the computation and ask if the formula is satisfiable.

A boolean expression which is satisfiable if and only if an arbitrary nondeterministic

Turing Machine accepts a string is given by:

$$\bigwedge_{t\in T}\bigwedge_{q\in Q} S_{t,q} \implies \bigwedge_{q'\in T\setminus\{q\}} \neg S_{t,q'}$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P} H_{t,p} \implies \bigwedge_{p'\in P\setminus\{p\}} \neg H_{t,p'}$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{\sigma\in\Sigma} T_{t,p,\sigma} \implies \bigwedge_{\sigma'\in\Sigma\setminus\{\sigma\}} \neg T_{t,p,\sigma'}$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{p'\in P\setminus\{p\}} (H_{t,p} \wedge T_{t,p',\sigma}) \implies T_{t+1,p',\sigma}$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{q\in Q} S_{t,q} \wedge T_{t,p} \wedge T_{t,p,\sigma} \implies \bigvee_{\Delta} S_{t+1,q'} \wedge H_{t+1,p'} \wedge T_{t+1,p,\sigma'}$$

$$S_{1,s} \wedge H_{1,1}$$

$$\bigwedge_{p\leq p'} T_{1,p,x_p} \wedge \bigwedge_{n<p} T_{1,p,\sqcup}$$

$$\bigvee_{t} S_{t,\text{acc}}$$

However, these formulae are not in conjunctive normal form. We can rewrite them (in polynomial time) such that they are (providing a reduction from SAT to CNF-SAT):

$$\bigwedge_{t\in T}\bigwedge_{q\in Q}\bigwedge_{q'\in T\setminus\{q\}} (\neg S_{t,q} \vee \neg S_{t,q'})$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{p'\in P\setminus\{p\}} (\neg H_{t,p} \vee \neg H_{t,p'})$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{\sigma\in\Sigma}\bigwedge_{\sigma'\in\Sigma\setminus\{\sigma\}} (\neg T_{t,p,\sigma} \vee \neg T_{t,p,\sigma'})$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{p'\in P\setminus\{p\}} \neg H_{t,p} \vee \neg T_{t,p',\sigma} \vee T_{t+1,p',\sigma}$$

$$\bigwedge_{t\in T}\bigwedge_{p\in P}\bigwedge_{q\in Q}\bigwedge_{\Delta} \neg S_{t,q} \vee \neg T_{t,p} \vee \neg T_{t,p,\sigma} \vee AUX_{t+1,q',p',\sigma'}$$

$$\bigwedge_{t\in T}\bigwedge_{q\in Q}\bigwedge_{p\in P}\bigwedge_{\sigma\in\Sigma} \neg AUX_{t,q,p,\sigma} \vee H_{t,p}$$

$$\bigwedge_{t\in T}\bigwedge_{q\in Q}\bigwedge_{p\in P}\bigwedge_{\sigma\in\Sigma} \neg AUX_{t,q,p,\sigma} \vee S_{t,q}$$

$$\bigwedge_{t\in T}\bigwedge_{q\in Q}\bigwedge_{p\in P}\bigwedge_{\sigma\in\Sigma} \neg AUX_{t,q,p,\sigma} \vee T_{t,p,\sigma}$$

$$S_{1,s} \wedge H_{1,1}$$

$$\bigwedge_{p\leq p'} T_{1,p,x_p} \wedge \bigwedge_{n<p} T_{1,p,\sqcup}$$

$$\bigvee_{t} S_{t,\text{acc}}$$

(b) Show that 3-SAT is NP-complete by means of a suitable reduction.

To prove that 3-SAT is NP-hard; I provide a reduction from CNF-SAT to a 3-SAT formula which is equisatisfiable. This satisfies the criteria for a polynomial time reduction. Given a CNF-SAT formula ϕ let ϕ' be the 3-SAT formula which is equisatisfiable.

Firstly, copy all the conjuncts containing 1, 2 or 3 literals into ϕ'.

Next, take all conjuncts containing ≥ 4 literals $A_1 \vee A_2 \vee \cdots \vee A_m$ and add $(A_1 \vee A_2 \vee n_1) \wedge (\overline{n_1} \vee A_3 \vee n_2) \wedge \cdots \wedge (\overline{n_{m-4}} \vee A_{m-1} \vee A_m)$ to ϕ'. This formula is satisfiable if and only if ϕ is satisfiable. Therefore, this is a valid reduction from CNF-SAT to 3-SAT.

3-SAT is clearly in NP – the algorithm which nondeterministically guesses an allocation and then tests whether this allocation satisfies the formula is a nondeterministic algorithm for 3-SAT.

(c) Give a polynomial time reduction from 3-SAT to 4-NAE. What can you conclude about the complexity of the latter problem?

I provide a reduction f from a formula ϕ in 3-CNF to a 4-CNF formula ψ which is in 4-NAE if and only if ϕ is in 3-SAT.

Define f as the function which adds a fresh literal X to all CNF clauses.

- Proof that $\phi \in 3-\mathsf{SAT} \implies f(\phi) \in 4-\mathsf{NAE}$

  Assume ϕ is satisfiable. Therefore there is some assignment A such that under assignment A, ϕ is satisfied.

  By applying $A + \{X \mapsto \mathsf{false}\}$ to $f(\phi)$, we have an interpretation where every clause contains at least one positive literal (since ϕ is satisfied by A) and at least one negative literal (since X is in every clause). This is the definition of *not-all-equal*.

  So $\phi \in 3-\mathsf{SAT} \implies f(\phi) \in 4-\mathsf{NAE}$ as required.

- Proof that $\phi \notin 3-\mathsf{SAT} \implies f(\phi) \notin 4-\mathsf{NAE}$

  Assume $\phi \notin 3-\mathsf{SAT}$, so there is no interpretation under which all clauses are satisfied.

  For all interpretations $A \setminus \{X\}$, we have that some clause is all negated. So for all clauses to have at least one positive literal, we require $X \mapsto \mathsf{true}$. If $f(\phi) \in 4-\mathsf{NAE}$ then we all clauses in $f(\phi)$ must have at least one negative literal. Since $X \mapsto \mathsf{true}$ (by above), we have that some literal in every clause in ϕ must be negated. Consider inverting the assignment A. This would result in an assignment where every clause in ϕ contains a literal which is positive. So $\phi \in 3-\mathsf{SAT}$. However, this contradicts the assumption that $\phi \notin 3-\mathsf{SAT}$.

  This proves that $\phi \notin 3-\mathsf{SAT} \implies f(\phi) \notin 4-\mathsf{NAE}$

(d) Show that the problem 3-NAE is NP-complete

I shall describe a reduction from 4-NAE to 3-NAE which shows that 3-NAE is NP-complete. This reduction shall take a similar structure to the reduction from SAT to 3-SAT.

Start with an expression in 4-NAE. For every clause $(A \vee B \vee C \vee D)$ containing exactly 4 literals, split it into two clauses: $(A \vee B \vee n_i) \wedge (\neg n_i \vee C \vee D)$ where n_i is a fresh variable which does not occur anywhere else in ϕ. The expression formed by this translation is in 3-NAE if and only if the original expression was in 4-NAE. Therefore this is a valid reduction from 4-NAE to 3-NAE.

- Assume that ϕ is in 4-NAE.

  So there exists an interpretation I under which every clause in I is satisfiable and not-all-equals. Let ψ be the translated expression.

  Since all clauses with 1–3 literals are unchanged, this interpretation means that all clauses with 1–3 literals are also in 3-NAE.

All clauses with 4-literals ($A \lor B \lor C \lor D$) are split into two clauses of the form $(A \lor B \lor n_i) \land (\neg n_i \lor C \lor D)$. Under the interpretation I, at least one of A, B, C, D is true and at least one is false.

If all expressions in the lhs clause are true then at least one expression in the rhs clause is false. So $n_1 \mapsto 0$ will result in both clauses being in 3-NAE. Similar logic applies if all expressions in the lhs clause are true, rhs clause are false and rhs clause are true. If none of these cases hold then both clauses are already in 3-NAE. Therefore, if ϕ is in 4-NAE then ψ is in 3-NAE.

- Assume that ϕ is not in 4-NAE

  So there exists some clause in ϕ which is not satisfiable and not-all-equals.

  If it has fewer than 4 literals, this clause is also in ψ so ψ is also unsatisfiable and not-all-equals.

  If this clause has 4 literals – if it is unsatisfiable, then the corresponding clauses in ψ are unsatisfiable. If it is all-true, then the resulting clauses are $(A \lor B \lor n_i) \land (\neg n_i \lor C \lor D)$. Therefore, all A, B, C and D hold. Whatever value n_i is assigned to, at least one of the clauses will be all equals.

  Therefore, ψ is not in 3-NAE if ψ is not in 4-NAE.

Since we have informally proved both directions, we can conclude that ψ is in 3-NAE if and only if ϕ is in 4-NAE – therefore the function described is a (polynomial-time) reduction. Since 4-NAE is a NP-complete problem, we can conclude that 3-NAE is also an NP-complete problem.

2 2015 Paper 6 Question 1

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2015p6q1.pdf

(b) An instance of a *linear programming* problem consists of a set $X = \{x_1, \dots, x_n\}$ of variables and a set of *integer constraints*,e ach of which is of the form

$$\sum_{1 \le i \le n} a_i x_i \le b,$$

where each a_i and b is an integer.

The $0-1$ Integer Linear Programming feasibility problem (ILP) is, to determine, given such a linear programming problem, whether there is an assignment of values from the set $\{0, 1\}$ to the variables in X such that substituting these values into the constraints leads to all constraints being simultaneously satisfied.

(i) Consider a *clause* c, i.e. a disjunction of Boolean literals. Show how such a clause can eb converted to an integer constraint which has a $\{0, 1\}$-solution if, and only if, c is satisfiable.

From a clause $\{X_1, X_2, \dots, X_n\}$ we can create an equisatisfiable linear programming problem of the form $\sum_{1 \le i \le n} a_i x_i \le b$, by taking $a_1 = a_2 = \dots a_n = b = -1$.

If c is satisfiable then there exists some interpretation such that at least one of the clauses is true. Consider all true variables to have value 1 and all false variables to have value 0. So if c is satisfied by some interpretation then $\sum_{1 \le i \le n} X_i \ge 1 \simeq \sum_{1 \le i \le n} -1 \cdot X_i \ge -1$. This second linear constraint is the mapping I defined above. So if c is satisfiable then the second ILP expression is satisfiable.

If the clause c is not satisfiable then for all interpretations, every variable in the clause is false. Considering all false variables to be 0, we have for all interpretations, $\sum_{1 \le i \le n} X_i = 0 \simeq \sum_{1 \le i \le n} -X_i = 0 \implies \sum_{1 \le i \le n} -X_i > -1$. Therefore the integer constraint is unsatisfiable.

So the integer constraint formed is satisfiable if and only if the *clause c* is satisfiable.

(ii) Use part (b)(i) to show that there is ap olynomial-time reduction from the problem $\mathsf{CNF-SAT}$ to ILP.

Define f as follows: take as input an expression ϕ in CNF and convert each clause into an integer constraint as described above. This is clearly polynomial.

If ϕ is satisfiable then there exists some interpretation $\mathcal{I}$ such that every clause is simultaneously satisfied. By (b)(i), we have that this implies all the integer constriants in $f(\phi)$ can be simultaneously satisfied. Therefore, the expression formed is in ILP.

If ϕ is not satisfiable then for every interpretation $\mathcal{I}$, we have that at least one clause is not satisfied. By (b)(i) we have that the integer constraint formed by this clause is also not satisfied. So for every interpretation, there is at least one integer constraint in $f(\phi)$ which is not satisfied. So $f(\phi)$ is not in ILP.

Therefore, f satisfies the required criteria to be a polynomial -time reduction from $\mathsf{CNF-SAT}$ to ILP:

$$x \in \mathsf{CNF-SAT} \iff f(x) \in \mathsf{ILP}$$

(iii) Is there a polynomial-time reduction from ILP to $\mathsf{CNF-SAT}$? Justify your answer.

There is a polynomial-time reduction from ILP to $\mathsf{CNF-SAT}$.

By the Cook-Levin theorem, $\mathsf{CNF-SAT}$ is NP-complete. By the definition of NP-completeness, this implies that $\forall A \in \mathsf{NP}. A \leq_P \mathsf{CNF-SAT}$.

So to prove that there is a reduction from ILP to $\mathsf{CNF-SAT}$, it suffices to prove that ILP is in NP.

ILP can be solved in polynomial time by a Nondeterministic Turing Machine by nondeterministically writing an assignment to all variables and (which takes polynomial time) and then testing whether all the integer constraints are simultaneously satisfied (which also takes polynomial time). Therefore, $\mathsf{ILP} \in \mathsf{NP}$. So by the Cook-Levin Theorem there exists a polynomial time reduction from ILP to $\mathsf{CNF-SAT}$.

(iv) What can you conclude about the complexity of ILP?

From (b)(ii), we have that $\mathsf{CNF-SAT} \leq_P \mathsf{ILP}$. Since $\mathsf{CNF-SAT}$ is NP-complete, we have that ILP is NP-hard.

From (b)(iii), we have that $\mathsf{ILP} \in \mathsf{NP}$.

Combining these (using the definition of NP-completeness from (a)), we have that ILP is NP-complete.

3 2014 Paper 6 Question 2

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2014p6q2.pdf

(b) Consider the following two decision problems.

Problem 1: Given an undirected graph $G = (V, E)$ with $|V|$ even, does G contain a clique with at least $|V|/2$ vertices?

Problem 2: Given an undirected graph $G = (V, E)$, does G contain a clique with at least $|V| - 3$ vertices?

(i) Which of the two problems is in P and which one is NP-complete?

Problem 1 is NP-complete.

Problem 2 is in P.

(ii) For the problem in P, describe a polynomial-time algorithm.

- For every triple of vertices (v_i, v_j, v_k), let E' be the cardinality of the set of edges which contain one of the three vertices. If $|E| - |E'| = (|V| - 3)!$ then succeed
- If no triple of vertices met the criteria then fail

There are $|V|^3$ vertices, at each of which we do $|E|$ work. Therefore, the complexity of this algorithm is $\mathcal{O}(|V|^3|E|)$. This proves that Problem $1 \in \mathsf{P}$.

(iii) For the other problem, prove that it is NP-complete.

I provide a reduction from CLIQUE to Problem 1. Since CLIQUE is NP-complete, this proves that Problem 1 is NP-hard.

I then show that Problem 1 is in NP.

Combining these results shows that Problem 1 is NP-complete.

The input to CLIQUE is a pair with a graph and an integer (G, k), where G itself is a pair (V, E). CLIQUE accepts if there is a clique with at least k nodes in the graph G. Define the reduction f as follows:

$$f(((V,E),k)) \triangleq \begin{cases} (V + \{v_1, \ldots, v_{2k-|V|}\}, E) & \text{if } k \geq \lceil |V|/2 \rceil \\ (V' = (V + \{v_1, \ldots, v_{|V|-2k}\}), E + \{(v_1, v), \ldots, (v_{|V|-2k}, v) | v \in V'\}) & \text{if } k < \lceil |V|/2 \rceil \end{cases}$$

Intuitively, if k is greater than $\lceil |V|/2 \rceil$, we add $2k - |V|$ nodes with no neighbours such that k is exactly half the number of nodes in the augmented graph and the size of the largest clique is unaffected. If k is less than half, then we add $|V| - 2k$ nodes which are connected to everything to increase the size of the maximum clique by $|V| - 2k$.

This is clearly polynomial-time computable.

I prove that f satisfies the other requirements of a polynomial-time reduction from IND to Problem 2:

$$x \in \mathsf{IND} \iff f(x) \in \text{Problem 2}$$

- $(\implies)$

  Assume $G \in \mathsf{IND}$

  Let $G' = f(G)$.

  - Case $k \geq |V|/2$

    If there is a clique in G of size k, then (since G' contains G as a subgraph) there must be a clique of size k in G'. By the way G' is constructed, we have $|V'| = 2k$ so there is a clique containing at least half the nodes in the graph G'.

    So we have $f(x) \in$ Problem 2 in this case.

  - Case $k < |V|/2$

    In this case, there is a clique of size k in G. f adds $|V| - 2k$ fully connected vertices into the graph. So this implies there is a clique of size $k + |V| - 2k = |V| - k$ in G'. G' has $2|V| - 2k$ vertices. So there is a clique containing at least half of the vertices in the graph.

    So $f(G) \in$ Problem 2 in this case.

- $(\Longleftarrow)$

  Assume $f(G) \in$ Problem 2.

  - Case $k \geq |V|/2$

    If $f(G) \in$ Problem 2, then there is a clique of size at least $|V'|/2$. By the way V' is constructed, we know $|V'| = 2k$. So there is a clique in G' of size at least k. f added $2k - |V|$ new nodes to the G which are unconnected. So these nodes must not appear in the clique of size k. So the clique of size k must be in the original graph G.

    So we have $G \in \mathsf{IND}$

  - Case $k < |V|/2$

    If $f(x) \in$ Problem 2 then there is a clique of size at least $|V'|/2$ in G'. This clique contains at least $|V| - k$ vertices. G' is G but with $|V| - 2k$ fully connected vertices. So at least k of these vertices must be from the original graph G. So these k vertices form a clique with each other. Hence the original graph G contains a clique with at least k vertices.

    So we have $G \in \mathsf{IND}$

This proves that $\mathsf{IND} \leq_P$ Problem 2. By the composition of polynomial-time reductions, we have that $\forall A \in \mathsf{NP}. A \leq_P$ Problem 2. So Problem 2 is NP-hard.

Problem 2 is clearly in NP. A nondeterministic Turing Machine can nondeterministically select a set of vertices of size $|V|/2$ and determine whether they form a clique in polynomial time.

Since Problem 2 is both NP-hard and in NP, we have that Problem 2 is NP-complete.