Introduction to Theoretical Computer Science

Lecture 12: NP-Completeness

Dr. Liam O'Connor

University of Edinburgh

Semester 1, 2023/2024

Hardness

Definition

A problem P_1 is *polynomially reducible* to P_2, written $P_1 \leq_P P_2$, if there is a polynomially-bounded reduction from P_1 to P_2.

Recall:
To prove that a problem P_2 is *hard*, show that there is an *easy* reduction from a known hard problem P_1 to P_2.

Definition

A problem P is ***NP-Hard*** if, for every $A \in \mathbf{NP}$, $A \leq_P P$

- If a problem P_1 is **NP**-hard and $P_1 \leq_P P_2$ then P_2 is **NP**-Hard.
- To prove that a problem P_2 is **NP**-hard, show that there's a polynomial reduction from a known **NP**-hard P_1 to P_2.

Completeness

Question

If any **NP**-hard problem is shown to be in **P**, what does that mean?

Definition

A problem is ***NP**-complete* if it is both **NP**-hard and in **NP**.

Do **NP**-Complete Problems Exist?

There are **many** such problems, including *HPP* and Timetabling. In fact, almost all **NP**-problems encountered in practice are either in **P**, or **NP**-complete.

Computers and Intractability - A guide to theory of NP-completeness, M.R. Garey and D.S. Johnson, Freeman 1979 lists a whole bunch.

The original **NP**-Complete problem

The *Cook-Levin theorem* states that a particular **NP** problem, SAT, is **NP**-complete. The theorem is usually proved for TMs; we shall do it later for RMs.

Why Cook-Levin?

The notion of **NP**-completeness, and the theorem, were due to Stephen Cook (and partly Richard Karp)—in the West. But as with many major mathematical results of the mid-20th century, they were discovered independently in the Soviet Union, by Leonid Levin. Since the fall of the Iron Curtain made Soviet maths more accessible, we try to attribute results to both discoverers.

SAT

SAT is a very significant problem about boolean formulae.

The SAT Problem

Given a boolean formula φ over a set of boolean variables X_i, is there an *assignment* of values to X_i which *satisfies* φ?

(i.e. makes φ true?)

Equivalently, is φ non-contradictory?

- $(A \vee B) \wedge (\neg B \vee C) \wedge (A \vee C)$ is satisfiable, e.g. by making A and C true. $(A \wedge B) \wedge (\neg B \wedge C) \wedge (A \wedge C)$ is not satisfiable.
- The size of a *SAT* problem is the number of symbols in φ.
- *SAT* is obviously in **NP**: just nondeterministically "guess" an assignment and check it.
- It's also apparently exponential in reality: no obvious way to avoid checking all possible assignments (the truth table method).

The original **NP**-Complete problem

The *Cook-Levin theorem* states that a particular **NP** problem, *SAT*, is **NP**-complete. The theorem is usually proved for TMs; we shall do it later for RMs.

Why Cook-Levin?

The notion of **NP**-completeness, and the theorem, were due to Stephen Cook (and partly Richard Karp)—in the West. But as with many major mathematical results of the mid-20th century, they were discovered independently in the Soviet Union, by Leonid Levin. Since the fall of the Iron Curtain made Soviet maths more accessible, we try to attribute results to both discoverers.

SAT

SAT is a very significant problem about boolean formulae.

The SAT Problem

Given a boolean formula φ over a set of boolean variables X_i, is there an *assignment* of values to X_i which *satisfies* φ?

(i.e. makes φ true?)

Equivalently, is φ non-contradictory?

- $(A \vee B) \wedge (\neg B \vee C) \wedge (A \vee C)$ is satisfiable, e.g. by making A and C true. $(A \wedge B) \wedge (\neg B \wedge C) \wedge (A \wedge C)$ is not satisfiable.
- The size of a *SAT* problem is the number of symbols in φ.
- *SAT* is obviously in **NP**: just nondeterministically "guess" an assignment and check it.
- It's also apparently exponential in reality: no obvious way to avoid checking all possible assignments (the truth table method).

The Proof

- **The** *SAT* **problem is in NP**: Nondeterministically guess an assignment and check it in polynomial time.
- **The** *SAT* **problem is NP-Hard**: Shown by reduction from any **NP** problem to *SAT*.

The Reduction

Suppose $(D, Q) \in$ **NP**. We shall construct a reduction $Q \leq_P SAT$. Given an instance $d \in D$, we shall construct a formula φ_d which can be satisfied if its variables describe the successful executions of an NRM checking Q. This machine can be polynomially bounded, so the size of φ_d will be polynomial in the size of d.

The Variables

Our NRM for Q, $M = (R_0, \ldots, R_{m-1}, I_0, \ldots, I_{n-1})$ runs for s steps (i.e. $p(|d|)$ where d is our input and p is our polynomial bound).

Name	**Meaning**	**How Many**
C_{tj}	Program counter at step t is on I_j.	$s \cdot n$
R_{tik}	kth bit of R_i at step t.	$s \cdot m \cdot 2s$

Why $2s$?

How big can the registers get? Running s steps of $\mathrm{ADD}(0, 0)$ will make R_0 double s times, if it starts at $2^{|d|}$ then we need $2^{|d|+s}$ capacity. Then w.l.o.g. 2^{2s} i.e. $2s$ bits is enough.

The Formula

$$C_{00} \wedge \rho_{\text{init}} \wedge \chi_{\text{one}} \wedge \bigwedge_t \chi_t \wedge \alpha$$

Name	**Meaning**	**How Many**
χ_{one}	Program counter is in one place.	$s \cdot n^2$
χ_t	Step $t+1$ follows from step t.	$s^2 \cdot m$
ρ_{init}	Initial register values	$m \cdot n$
α	machine accepts	s

Details

Some formulae are easy:

Program counter is in one place

$\chi_{\text{one}} \equiv \bigwedge_t \bigvee_j \left(C_{tj} \wedge \bigwedge_{j' \neq j} \neg C_{tj'} \right)$

Some are more tedious:

Step $t+1$ follows from step t

$\chi_t \equiv \varphi_t \wedge \rho_t$ where φ_t models control flow changes and ρ_t models register changes.

$\varphi_t \equiv \bigvee_j (C_{tj} \wedge \nu_{tj})$, where ν_{tj} is:

- $C_{t+1,j+1}$ if I_j is INC, ADD, or SUB.
- $C_{t+1,j+1} \vee C_{t+1,j'}$ if I_j is MAYBE(j')
- $((\bigvee_k R_{tik}) \wedge C_{t+1,j+1}) \wedge ((\bigwedge_k \neg R_{tik}) \wedge C_{t+1,j'})$ if I_j is DECJZ(i, j')

Register changes

The formulae concerning registers are very tedious, but can easily be found. See your hardware course!

(Also Julian's old notes for this course, which someone should remind me to upload)

Exercise

Write a formula $\rho^{+}_{tii'}$ which states that at step $t+1$, R_i will have the sum of the values in R_i and $R_{i'}$ at step t.

Despite this being very tedious, these formulae are polynomial ($\mathcal{O}(s^4)$) !

3SAT

- φ is in *conjunctive normal form* if it is of the form $\bigwedge_i \bigvee_j P_{ij}$ where each P_{ij} is a *literal* (either a variable P or negation of one $\neg P$.).
- φ is in *k-CNF* if each clause $\bigvee_j P_{ij}$ has at most k literals.

The Problem

3SAT is the problem of whether a satisfying assignment exists for a formula in 3-CNF.

Reduction from *SAT* to *3SAT* is difficult because normally converting to 3-CNF is an exponential blowup. The *Tseitin encoding* is used instead to give a not-equivalent but *equisatisfiable* formula (See Julian's note, attached).

Note: Cook-Levin for TMs is already for 3SAT.

Clique

The *CLIQUE* problem

Given a graph $G = (V, E)$ and a number k, a *k-clique* is a k-sized subset C of V, such that every vertex in C has an edge to every other. (C forms a complete subgraph.) Decide whether G has a k-clique.**Exercise**: Why is *CLIQUE* $\in$ **NP**?

Reducing from *3SAT*, we have a formula

$$\varphi = \bigwedge_{1 \leq i \leq k} (x_{i1} \vee x_{i2} \vee x_{i3})$$

The Graph

Each x_{ij} is a vertex. Connect x_{ij} to $x_{i'j'}$ iff: $i \neq i'$ and $x_{i'j'}$ is not the negation of x_{ij}.

i.e. we connect literals in different clauses so long as they are not inconsistent.

Why does this work?

Since the vertices in one clause are disconnected, finding a k-clique amounts to finding one literal for each clause, such that they are all consistent — and so represent a satisfying assignment. Conversely, any satisfying assignment generates a k-clique.

P vs. NP

As previously mentioned, we **don't know** if **P** and **NP** are really distinct classes.

Find a polynomial time algorithm for any **NP**-hard problem and you can win yourself one million US dollars from the Clay Institute. (Also hire bodyguards because most web/banking security depends on such problems being hard)

Many complexity theory results start with "*if* $\mathbf{P} \neq \mathbf{NP}$..."

Example (**NP**-Intermediacy)

A problem is **NP**-Intermediate if it is in **NP** but not in **P** nor **NP**-complete.

If $\mathbf{P} \neq \mathbf{NP}$, then graph isomorphism is such a problem (and there aren't many others).

NP in Practice

As far as we know, **NP** problems are just hard: need exponential search, so $\mathcal{O}(p(n) \cdot 2n)$. So how do we solve them in practice?

- Randomised algorithms are often useful. Allow algorithms to toss a coin. Surprisingly one can get randomised algorithms that solve e.g. *3SAT* in time $\mathcal{O}(p(n) \cdot \frac{4}{3}^{n})$.

  (Why is this useful? $2^{100} \approx 10^{31}$, while $1.33^{100} \approx 10^{12}$)

  Catch: (really) small probability of error!

- In many special classes (e.g. sparse graphs, or almost-complete graphs), heuristics lead to fast results. See `http://satcompetition.org/` for the state of the art.

Next time...

We'll be looking at the boundaries of the class **NP**, and what lays beyond. Specifically, the classes of **coNP** and **PSPACE**, as well as the *polynomial hierarchy*, analogous to the arithmetic hierarchy we've already seen, but contained entirely within **PSPACE** decidable problems.

If time, I might mention the *sublinear* classes of **L** and **NL** as well, but these are not examinable.