NP-completeness

Evgenij Thorstensen

V18

Recap completeness

Recall $A \leqslant_P B$ if there is a polynomial-time many-one reduction from A to B.

Problems/strings in A can be transformed *correctly* into those of B in polynomial time.

A problem A is hard for a class (say NP) if $B \leqslant_P A$ for every $B \in \mathsf{NP}$.

A is complete for a class if it is both hard for this class and in it.

Usefulness of complete problems

The complete problems are the hardest, and they are universal.

Solvers for them can be used for all other problems.

Allow for attacks on inclusions and separations of classes.

To prove $\mathsf{P} = \mathsf{NP}$, sufficient to find a polynomial time algorithm for any single NP-complete problem.

NP-completeness

Theorem (Cook-Levin)

SAT is NP*-complete.*

The proof comes in two parts. One part is a clever idea, the second part is a lot of technical pizzlecacky.

The clever idea

Instead of transforming *problems* of NP into SAT instances, transform the NTM deciding them.

If I can do this in polynomial time, I get a universal reduction.

The resulting SAT instance will bear no resemblance to the original problem instance, but oh well.

Preliminaries

Let L be an arbitrary language in NP. Let M be the NTM deciding it in time $p(n)$.

We have

- States Q
- Alphabet Σ, assume $\{0, 1, \#\}$.
- Transitions Δ, of the form $Read(S, v) \rightarrow Move(S', v', \{-1, +1\})$
- Accept and reject states S_a and S_r

We will simulate M for up to $p(n)$ steps. To do so, let's number the tape cells.

Variables

I have variables tracking the tape, the head, and the state of M at each step.

- $T(i, j, k)$: Tape(CellNumber, Symbol, StepNumber)
- $Q(i, k)$: State(State, StepNumber)
- $H(i, k)$: Head(CellNumber, StepNumber)

How many of each?

Variables

I have variables tracking the tape, the head, and the state of M at each step.

- $T(i, j, k)$: Tape(CellNumber, Symbol, StepNumber)
- $Q(i, k)$: State(State, StepNumber)
- $H(i, k)$: Head(CellNumber, StepNumber)

How many of each?

- $T(i, j, k)$: $p(n) \times p(n) \times |\Sigma|$
- $Q(i, k)$: $p(n) \times |Q|$
- $H(i, k)$: $p(n) \times p(n)$

Some constraints

At most one symbol per cell per step:

$$\neg(T(i,j,k) \wedge T(i,j',k))$$

At most one state per step:

$$\neg(Q(i,k) \wedge Q(i',k))$$

At most one head position per step:

$$\neg(H(i,k) \wedge H(i',k))$$

Oh, and at least one symbol per cell

$$\bigvee_{j \in \Sigma} T(i,j,k)$$

More constraints

We want to forbid unauthorized assignments.

No magic writes!

$$T(i, j, k) \wedge T(i, j', k+1) \rightarrow H(i, k)$$

Will also need “no illegal transitions”; This is missing in the wikipedia proof and many other places.

Transitions

We want to force legal moves

$$H(i,k) \wedge Q(q,k) \wedge T(i,s,k) \rightarrow H(i+d,k+1) \wedge Q(q',k+1) \wedge T(i,s',k+1)$$

as a disjunction over all $Read(q,s) \rightarrow Move(q',s',d)$

We also need to ban illegal moves

$$(H(i,k) \wedge Q(q,k) \wedge T(i,s,k) \rightarrow \neg(H(i+d,k+1) \wedge Q(q',k+1) \wedge T(i,s',k+1))$$

as a conjunction for each missing transition with $Read(q,s)$.

Init and stop

After at most $p(n)$ steps, we need the accepting state.

$$\bigvee Q(a, k)$$

And initially, we have

$$T(i, j, 0)$$

and

$$Q(s, 0) \wedge H(0, 0)$$

Correctness

Correctness by construction — only legal transitions are allowed to make things true.

Multiple legal transitions allowed, but that's fine.

If we have a satisfying assignment, it must

- Make $Q(a, k)$ true for some k
- Make the starting conditions true
- Not violate any rule forbidding transitions — and we are forcing the legal ones.

Complexity

We made multiple sets of $O(p(n)^2)$ formulas.

So the resulting formula is of size $O(p(n)^2)$.

Cook-Levin, summarized

This is an adaptation of the proof on wikipedia.

https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem

Many proofs along these lines are missing formulas forbidding illegal transitions; beware.

Making more NP-complete problems

Now that we have SAT, can reduce it to a problem to prove completeness.

Helpful to have a restricted form of SAT, namely 3SAT.

$$\bigwedge (X \vee Y \vee Z)$$

where X, Y, Z are literals: Variables or negations of variables. This is CNF.

CNFSAT is NP-complete

Membership obvious.

For hardness, we can modify the Cook-Levin reduction to be in CNF.

To go from CNFSAT to 3SAT, we apply the following tricks:

A clause $(x_1 \vee x_2 \vee x_3 \vee x_4)$ can be split in half to yield

$$(x_1 \vee x_2 \vee z) \text{ and } (\bar{z} \vee x_3 \vee x_4)$$

Clauses that are too small can be padded: $x_1 \vee x_2$ becomes $x_1 \vee x_2 \vee x_2$

3SAT to CLIQUE

A clique is a complete undirected graph.

Problem (Clique)

Given a graph G *and* $k \in \mathbb{N}$, *does* G *contain a clique of size* k*?*

This reduction is in Sipser.

3SAT to clique

We let the occurences of literals in the 3SAT formula be our vertices.

We connect everything except:

- Pairs $x, \bar{x}$, and
- Occurences in the same clause

$$(x_1 \vee x_1 \vee x_2), (\bar{x}_1, x_2, x_2), (\bar{x}_1, \bar{x}_2, \bar{x}_2)$$

Proof

Let k be the number of clauses in the 3SAT formula ϕ. The resulting graph has a k-clique if and only if ϕ is satisfiable.

If I have such a clique, it must contain k nodes. There can only be one from each clause, and no two conflicting ones.

If I have a satisfying assignment, it assigns true to at least one literal from every clause, and no two conflicting ones. Take these, and observe that they from a clique.

3SAT to 3COL

This requires a bit of ingenuity.

We will use logic gates for the clauses, and a special gadget for the assignments.

Base gadget: A triangle with 3 nodes labelled T, F, and B.

For each variable, I make two nodes x and $\bar{x}$, and connect them to node B and to each other.

Gadgets

This node gadget tracks valid assignments.

Next, we need to make an OR gate. It will have three vertices, two inputs and one output, in a triangle. We connect clause literals to the inputs.

It's a lossy gadget, but it has the relevant property that

- If the nodes connected to inputs are both coloured F, the output node must be F.
- If one node is coloured T, there exists a colouring where the output is also T.

Almost done

A clause gadget has two OR gates. We connect the output of the second gate to nodes B and F in the variables gadget.

Done! If I have a satisfying assignment, I can colour the corresponding variables T (as in, same colour as the T node). Then I have a colouring giving T to the OR gate outputs.

For the other direction, suppose the graph i 3-colourable. Make an assignment by setting variable coloured T to true.

If this assignment is not satisfying, then there is a clause that evaluates to false. But then, I have some clause in my graph where all three elements are coloured F, and then my output node is F.

A word of warning

Very similar-looking problems may not be so similar after all.

Problem (Vertex cover (NP-c))

Given an undirected graph G *and* $k \in \mathbb{N}$*, does* G *have a vertex cover of size* $\leqslant k$*?*

Problem (Edge cover (P))

Given an undirected graph G *and* $k \in \mathbb{N}$*, does* G *have an edge cover of size* $\leqslant k$*?*

NP-completeness, summary

The complete problems for a class are the hardest.

Direct reductions chain; look for a known NP-complete problem that is similar to the one you have.

A lot of problems with parameters become polynomial-time solvable if the parameter is fixed.