Introduction to Theoretical Computer Science

Note 3 – reductions for NP-completeness

This note gives more details on the two additional NP-complete problems in lectures.

1 Reducing SAT to 3-SAT

Suppose we have a boolean formula ϕ over variables $X_1, \ldots, X_n$. We want to construct a 3-CNF formula ψ such that ψ is satisfiable iff ϕ is satisfiable.

We'll do this by introducing a number of additional variables which represent the subformulae of ϕ. Suppose that ϕ_0 is a subformula of ϕ, and that ϕ_0 has the form $\phi_1 \wedge \phi_2$. We introduce new variables Y_0, Y_1, Y_2 representing the ϕ_i. We want $Y_0 = Y_1 \wedge Y_2$ – the key observation is that we can re-state this by saying that

$$(Y_0 \wedge Y_1 \wedge Y_2) \vee (\overline{Y_0} \wedge (\overline{Y_1} \vee \overline{Y_2}))$$

which we can then re-write (by brute force) into CNF:

$$\begin{aligned} &\big((Y_0 \vee \overline{Y_0}) \wedge (Y_0 \vee (\overline{Y_1} \vee \overline{Y_2}))\big) \\ \wedge &\big((Y_1 \vee \overline{Y_0}) \wedge (Y_1 \vee (\overline{Y_1} \vee \overline{Y_2}))\big) \\ \wedge &\big((Y_2 \vee \overline{Y_0}) \wedge (Y_2 \vee (\overline{Y_1} \vee \overline{Y_2}))\big) \end{aligned}$$

which simplifies down to

$$(Y_0 \vee \overline{Y_1} \vee \overline{Y_2}) \wedge (Y_1 \vee \overline{Y_0}) \wedge (Y_2 \vee \overline{Y_0})$$

which, we observe, is a formula in 3-CNF.

We can do a similar construction if ϕ_0 has the form $\phi_1 \vee \phi_2$ (**exercise:** do it).

In the event that ϕ_1 (or ϕ_2) is a literal p (i.e. is X_i or $\overline{X_i}$, we can just use p instead of introducing a new variable Y_1.

To produce our equisatisfiable formula ψ, we simply take the conjunction of all these formulae, together with the single conjunct Y, where Y is the variable corresponding to ϕ itself. By our construction, any assigment to the Xs and Ys that satisfies ψ, gives an assignment to Xs that satisfies ϕ; and conversely, given a ϕ-satisfying assignment to the Xs, we can construct a ψ-satisfying assignment to the Ys simply by giving each Y_i the value computed from its subformulae.

Note that while ψ is bigger than ϕ, it's only a small constant factor bigger (if ϕ has size n, ψ has size around $12n$).

2 Reducing SAT to CLIQUE

The proof on the slides is a proof, but it's a bit terse. Here it is again, fleshed out a bit.

Let $\phi = \bigwedge_{1 \le i \le k} (x_{i1} \vee x_{i2} \vee x_{i3})$ be an instance of 3SAT, so each x_{ij} is a literal over a set of variables $X_1, \ldots, X_n$.

We construct an instance of CLIQUE thus: let G be a graph, where the vertices are all the x_{ij} (for $1 \le i \le k$ and $1 \le j \le 3$), and there is an edge $(x_{ij}, x_{i'j'}$ iff $i \neq i'$ **and** the literal $x_{i'j'}$ is not the negation of the literal x_{ij}.

Thus we draw an edge between two literals exactly when they appear in different clauses and are not inconsistent with each other,

Obviously this is a polynomial construction.

Now, suppose ϕ is satisfiable, with some given satisfying assignment. That means that (at least) one literal in every clause $(x_{i1} \vee x_{i2} \vee x_{i3})$ must be true: choose one in each clause. These k literals are all consistent (since they derive from an assigment), and they're in different clauses, so there's an edge between any two of them, thus giving a k-clique in G.

Conversely, suppose G has a k-clique C. Since vertices are only joined if they're in different clauses, C must have one literal from each clause. All these literals are consistent (as they have edges between them), so we can get a satisfying assignment for ϕ just by making those literals in C true.