CSE491/596 Lecture Mon. Oct. 26: NP-Completeness

The definition of a language B being NP-complete is the same as before: $B \in$ NP and for all $A \in$ NP, $A \leq_m^p B$. All NP-complete decision problems are related by polynomial-time mapping equivalence, $\equiv_m^p$. Up at the top of NP (and hence also the top of co-NP) we will get a lot of more meaningful reduction equivalence thanks to completeness. Before tackling Cook's Theorem on the NP-completeness of SAT, let's see some simpler examples. Consider these decision problems:

CLIQUE

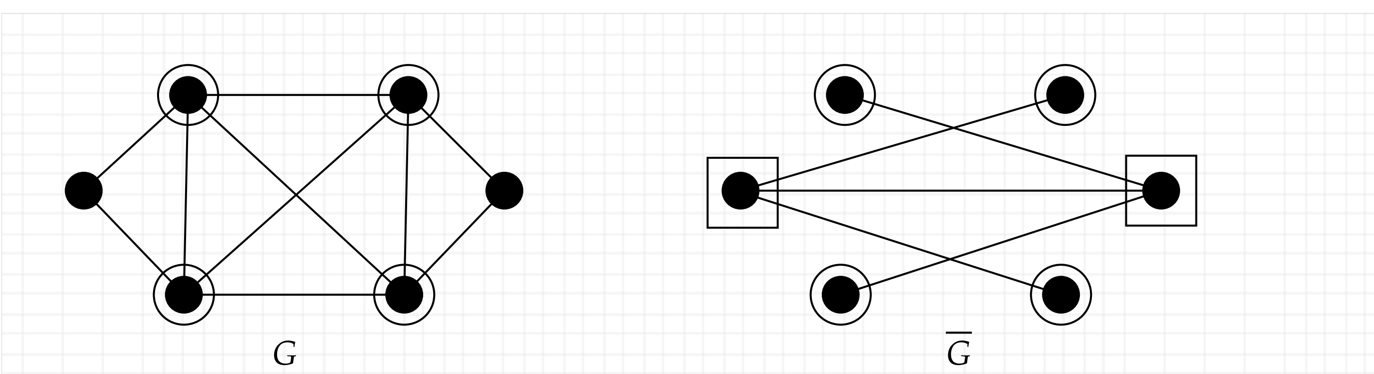
Instance: An undirected graph $G = (V, E)$ and a number $k \geq 1$.
Question: Does there exist a set $S \subseteq V$ of k (or more) nodes such that for each pair $u, v \in S$, (u, v) is an edge in E?

INDEPENDENT SET

Instance: An undirected graph $G = (V, E)$ and a number $k \geq 1$.
Question: Does there exist a set $S \subseteq V$ of k (or more) nodes such that for each pair $u, v \in S$, (u, v) is **not** an edge in E?

Important to keep straight: The languages of these problems are *not* complements of each other, despite their differing by just the word "not" at the end. Both languages are in NP with S as the witness. An important point is that with $n = |V|$, there are 2^n subsets S that might have to be considered. A polynomial-time algorithm cannot try each one. Within S, however, there are at most n^2 pairs (u, v) that have to be considered. Those can all be iterated through to check the body of the condition in quadratic time, so it becomes a polynomial-time decidable predicate $R(G, S)$. It is not even true that this predicate gets negated between the two languages, because it includes the "for each" part. It is because this runs over only polynomially-many pairs that I suggest the convention of saying "for each" rather than "for all" there. What actually gets complemented *is the graph* G, as expressed by this fact:

G has a clique of size $k \iff$ the complementary graph $\overline{G}$ has an independent set of size k.



Therefore, the simple reduction function $f(G, k) = (\overline{G}, k)$ reduces CLIQUE to IND SET and also viceversa, so the problems are $\equiv_m^p$ equivalent. [Note that this skips writing the angle brackets around $\langle G,k\rangle$; by now that's AOK.] A second fact yields a second equivalence:

The complement of an independent set S in G is a set S' of nodes such that every edge involves a node in S'. Such an S' is called (somewhat misleadingly, IMHO) a **vertex cover**. Therefore:

G has an independent set of size (at least) $k \iff G$ has a vertex cover of size (at most) $n-k$.

Note that the graph G stays the same; instead we flip around the target number from k nodes to $|V|-k$ nodes. In practice, when we're trying to optimize, we want to *maximize* cliques and independent sets and *minimize* vertex covers. The latter gives rise to this decision problem:

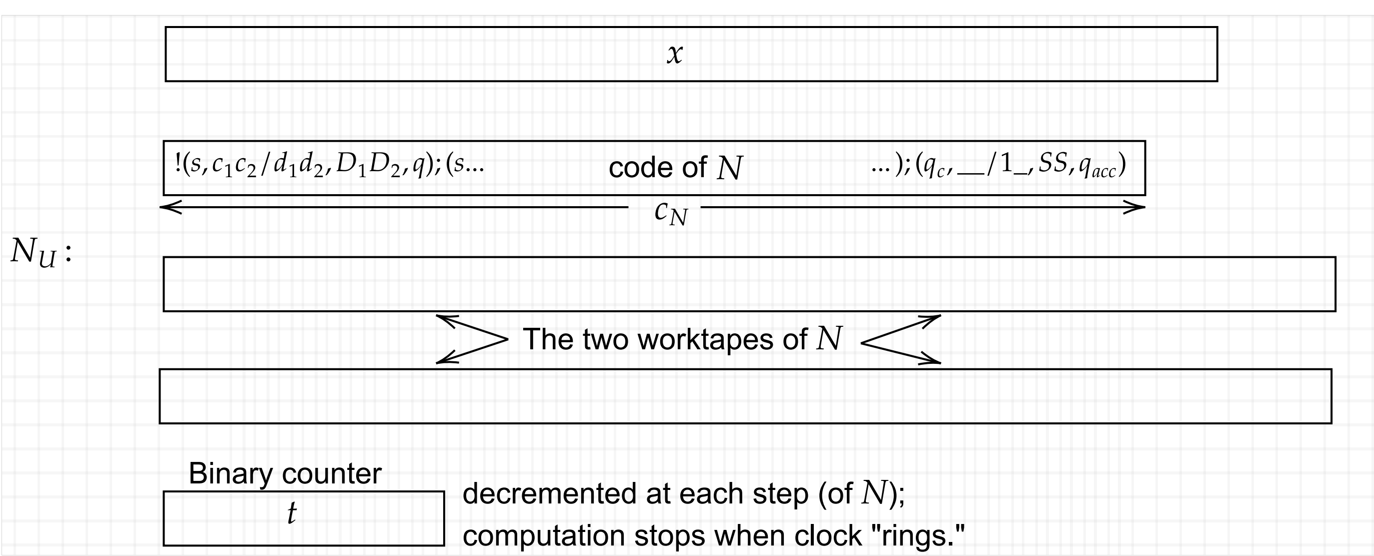
VERTEX COVER (VC)
Instance: A graph G and a number $\ell \geq 1$.
Question: Does G have a vertex cover of size (at most) ℓ?

Then IND SET and VC reduce to each other via the reduction $g(G,k) = (G,n-k)$ (where it is understood that $G = (V,E)$ and $n = |V|$.)

Next, we observe that NP has a complete set akin to A_{TM} but with an extra third component dedicated to balancing out the time complexity:

$$K_{NP} = \left\{\langle N, x, @^t\rangle : \textit{the 2-worktape NTM } N \textit{ accepts } x \textit{ within } t \textit{ steps}\right\}.$$

The reason this is in NP involves an important picture. We draw a 5-tape universal NTM N_U as follows. After N_U "unpacks" the three components of its input $z = \langle N, x, @^t\rangle$ onto its own tapes, the computation starts up looking like this:



The key point (which will matter more when we hit the Time Hierarchy Theorem) is that for N_U to execute the next step of N may require going thru its entire code of length c_N just to find the next applicable instruction. This is true all the more when the choice of the next instruction to execute is nondeterministic. Thus N_U does t steps of $N(x)$ in up to $c_N\, t$ steps of its own. In terms of the input x to N, c_N is a constant, but in terms of the input $z = \langle N, x, @^t \rangle$, which has length order-of $c_N + n + t$, the time $c_N\, t$ is quadratic in $r = |z|$. But that is completely fine: it puts K_{NP} into NTIME$\left[O\left(r^2\right)\right]$ which is within NP.

We have given N_U five tapes, one input tape and four worktapes, which may seem unfair. But we can invoke a general theorem, whose first part has been mentioned (but not proved) before. Its second part is complicated---both Debray and the ALR notes skip it (in our case, because it was included in someone else's chapter) and we will skip the proof here as well.

Theorem. For any multi-tape DTM (respectively, NTM) M that runs in time $t(n)$ and space $s(n)$, we can build:

1. a one-tape DTM (respectively, NTM) M_1 that simulates M in time $O\left(t(n)^2\right)$ and space $s(n)$;
2. a two-worktape DTM (respectively, NTM) M_2 that simulates M in time $O(t(n) \log t(n))$ and space $s(n)$.

Moreover, both M_1 and M_2 have the property that the location of their tape head(s) at any timestep t is a function of the length n of the input x alone, not of the content of x (this property is called "obliviousness"). ☒

What this means for any language $A \in$ P is that if we have a multi-tape TM M accepting A in polynomial time $t(n) = O\left(n^k\right)$, then we can get a 2-worktape TM M_2 that accepts A in time $O(t(n) \log t(n)) = O\left(n^k \log n\right) = \widetilde{O}\left(n^k\right)$. Likewise, given $A \in$ NP we may always take a 2-tape NTM N_2 to accept A in polynomial time; if A is in NTIME$\left[O\left(n^k\right)\right]$ then N_2 runs in time $\widetilde{O}\left(n^k\right)$ time (which we can bump up to time $O\left(n^{k+1}\right)$ if we don't like the tilde). We could even use a 1-tape NTM N_1 if we didn't care about doubling the exponent to time $O\left(n^{2k}\right)$. Now finally we can see why our language K_{NP} is NP-hard as well as belonging to NP.

Theorem. K_{NP} is NP-complete.

Proof. We have shown that K_{NP} is in NP. Now let any langauge $A \in$ NP be given. Then we can take a 2-worktape NTM N_A that accepts A in cn^k time for some constant c and exponent k. For any string x in Σ^* define

$$f(x) = \langle N_A, x, @^r \rangle \quad \text{where} \quad r = c|x|^k.$$

Then the function f is computable in deterministic time $O\left(n^k\right)$, most of which is spent writing down all the @ signs. Clearly $x \in A \iff N_A$ accepts x within r steps $\iff f(x) \in K_{NP}$, so f mapping-reduces A to K_{NP} in polynomial time.

Scholium (meaning, more important than a footnote in relation to course themes): The reduction of an arbitrary c.e. language A to AP_{TM} was $f(x) = \langle M_A, x\rangle$. This qualifies as a regular reduction because the "$\langle M_A,$" part is just a fixed string that can be output in arcs from the start state of a finite-state transducer T, and the final "$\rangle$" (or whatever concrete tuple-forming chars are used in its place) can be output using the final-state $\phi(q)$ function feature of the definition of an FST given on the presentation-options section of HW4. The reduction $f(x) = \langle x, x\rangle$ that we originally used from K_{TM} to A_{TM} is not regular, because of how it doubles x side-by-side. But we can instead take a fixed DTM M_K accepting the K_{TM} language and use M_K in place of M_A above, getting a regular reduction from K_{TM} to A_{TM} after all. In the reduction $f(G, k) = (\overline{G}, k)$ between CLIQUE and IND SET, if we represent the graph as a bitstring of $\binom{n}{2}$ edges and non-edges, then we need only complement this bitstring, which an FST can do. In the reduction $g(G, k) = (G, n-k)$ from IND SET to VC, we could argue the subtraction as doable by a multi-tape FST as in (D) from the first set of presentation options on HW2. We could also argue that by adding extra unused nodes we can make n a power of 2 minus 1 in the problem statement. Then $n-k$ becomes the same as complementing the binary expansion of k (aside from its leading 1), and that is regular without needing extra tapes.

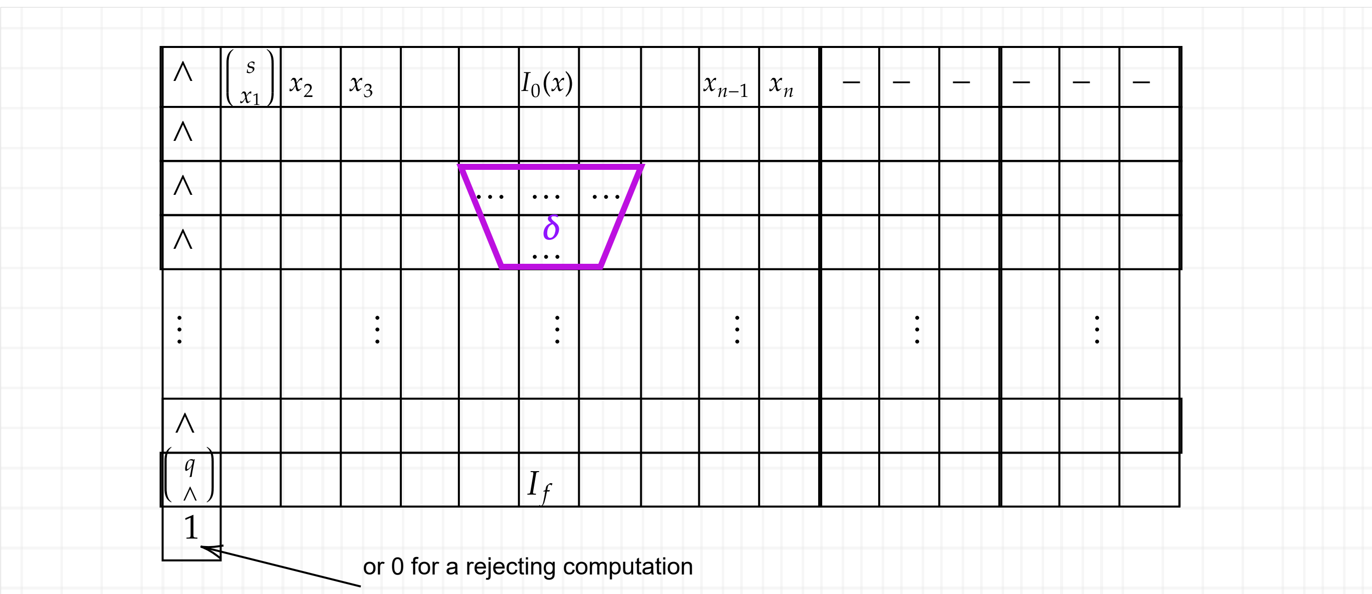
The above reduction to K_{NP} is not regular, however, for a firmer reason: because the final string $@^r$ requires counting up to the length of x. For this and similar reasons, the study of ``micro-reductions" has gone in two other directions:

- The logical notion of "projection" focuses on how x gets embedded one or more times into $f(x)$, so that going in the reverse direction, x can be "projected out of" $f(x)$.
- Allow $O(\log n)$ overhead to count with and do arithmetic on $O(\log n)$-sized binary numbers. Note that for any fixed k, you can count up to $r = cn^k$ in binary using numbers of size $\log c + k \log n = O(\log n)$. This is in keeping with allowing $O(\log n)$ bandwidth to "streaming algorithms"; before streaming algorithms came along we talked about $O(\log n)$ time for operations with random access---which is what "DLOGTIME" refers to in the ALR notes, *but you can skim/skip that aspect*.

And you can skim/skip this whole note, but it might feed into a later presentation option.

Thus the presence of complete languages in NP should not be a surprise, based on our experience with RE. The impact of the Cook-Levin Theorem---and the subsequent extension of completeness to CLIQUE and IND SET and VC and numerous other problems that had already been studied individually for decades---is that completeness holds for *natural* problems in NP. Indeed, we will see that all but a handful of the thousands of problems in NP have been classified either as in P or as NP-complete. (FACT is one of the few that stull have "intermediate" status.)

Before we state and prove the theorem, let us see one more application of the idea of tracing a sequence of IDs $I_0(x),\ I_1,\ I_2,\ \ldots,\ I_t$ that represent a valid t-step computation by a TM M, in this case a DTM. Whereas the Kleene T-predicate pictures them side-by-side, now we will stack them up into $t+1$ columns in a grid. For visual convenience we will suppose M is a 1-tape TM whose tape has a left end and is infinite only to the right, but this is not essential and we could add another grid to handle a second tape, with wires between the grids as well as within them. But for polynomial time, the simple one-plane grid is enough. Initially it has $n+1$ columns to hold the $\wedge$ left-endmarker and the input x. Over t steps, M cannot possibly visit more than t more cells, so we can lay the whole thing out on a $(t+1) \times s$ grid with $s \leq t+1$.



Every cell contains either a character in the work alphabet Γ of M or a pair in $Q \times \Gamma$ of a state and a char. We can use a binary encoding (a-la ASCII) of both. Then we can program a fixed finite function in Boolean logic, depending only on the instructions δ of M, that determines the contents of a cell in any row $i \geq 1$ depending only on the contents of it and its neighbor cell(s) in row $i-1$ for the previous timestep. The top row is initialized to $I_0(x)$ plus blanks to fill out the remaining columns up to t.

Because NAND is a universal gate, we can program the entire grid into a Boolean circuit C_x entirely of NAND gates, with an output wire w_0 at the bottom giving the final results, 1 or 0. Because the formula for δ over every cell is the same, the circuit C_x has such a regular structure (pun quasi-intended) that it is easily computed in $O(t^2)$ time given x. [Added afterward] The "x" is used only once and the values of its bits do not affect the layout, so we can give it via n **input gates** to what is otherwise a circuit C_n that depends only on the length n of x. We could suppose $\Sigma = \{0,1\}$ so x is already in binary, but we could also regard the Boolean encoding of $\Gamma \cup (Q \times \Gamma)$ that the circuit is already using as implicit at the inputs, so there are really $n' = O(n)$ binary input gates. The theorem we have proved has its own significance:

Theorem (often attributed to John E. Savage): For any language A in P and all n we can compute in $n^{O(1)}$ time a circuit C_n of NAND gates such that for all $x \in \Sigma^n$, $x \in A \iff C_n(x) = 1$. ⊠

The meaning of this theorem is that "software can be burned into hardware." The fact that $f(x) = \langle C_n, x \rangle$ is polynomial-time computable goes into saying that the sequence $[C_n]_{n=1}^{\infty}$ of circuits is **P-uniform**. The only reason f is not a "regular reduction" just like the reduction to A_{TM} is that C_n needs counting up to $n = |x|$ and more, and FSTs like DFAs cannot do unbounded counting. But it is close-to-regular in other senses of the above "Scholium" that in fact we get the stronger notion of being **DLOGTIME-uniform**.

Similar diagram from the ALR notes, ch. 27, section 3, showing how each cell depends on its 3 neighbors in the previous row:

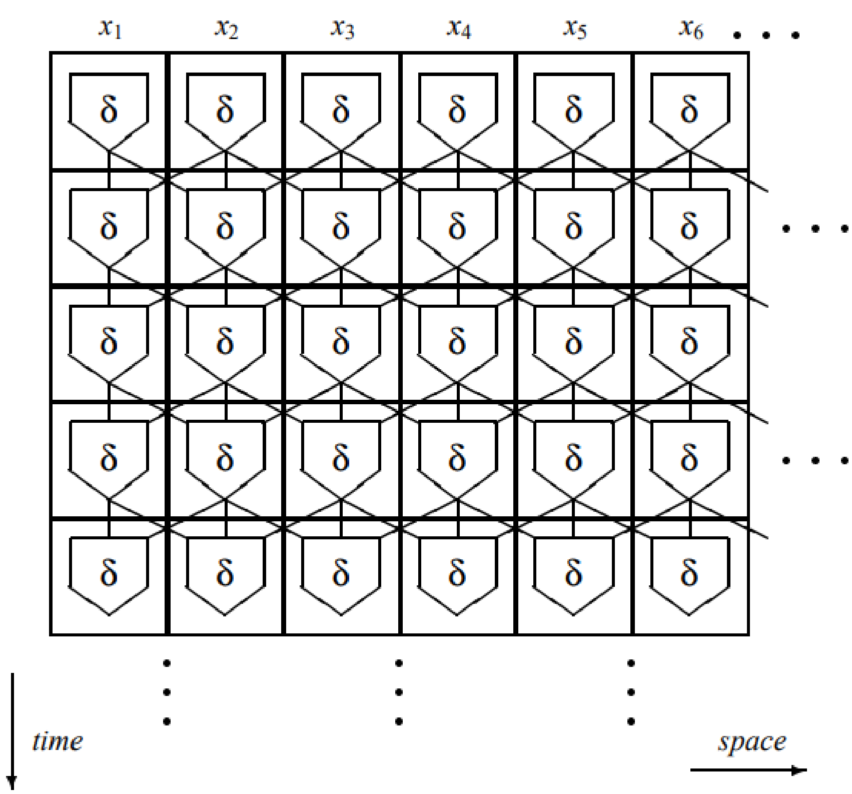


Figure 1: Conversion from Turing machine to Boolean circuits

To come on Wednesday: the proof of Theorem 3.1 in ALR ch. 28, now called the Cook-Levin Theorem.