CSE491/596 Lecture Fri. 10/20: NP-Completeness Proofs

First some remarks on the Cook-Levin Theorem: The mapping f from instance strings x of the general NP-language A produced a formula ϕ_x with clauses of 1, 2, or 3 literals. We can make all clauses have length exactly 3, and with no repeated variables in a clause, by the following trick: The formula

$$(u \vee v \ \vee \bar{z}) \ \wedge \ (\bar{u} \vee v \ \vee \bar{z}) \ \wedge \ (u \vee \bar{v} \ \vee \bar{z}) \ \wedge \ (\bar{u} \vee \bar{v} \ \vee \bar{z})$$

can be satisfied only by making z false. We can conjoin it to ϕ_x and do likewise with

$$(u' \vee v' \ \vee \bar{z}') \ \wedge \ (\bar{u}' \vee v' \ \vee \bar{z}') \ \wedge \ (u' \vee \bar{v}' \ \vee \bar{z}') \ \wedge \ (\bar{u}' \vee \bar{v}' \vee \bar{z}')$$

Then insert z into each clause with 2 variables and add z' for the 1-clauses, e.g., changing the output clause (w_0) to $(w_0 \vee z \vee z')$. Then the resulting ϕ'_x is in **strict 3CNF** and is likewise satisfiable if and only if $x \in A$.

Note: if, say, $x \ = \ 10110$, then ϕ_x can have the singleton clauses

$$(x_1 \vee z \vee z') \wedge (\bar{x}_2 \vee z \vee z') \wedge (x_3 \vee z \vee z') \wedge (x_4 \vee z \vee z') \wedge (\bar{x}_5 \vee z \vee z')$$

The only thing keeping f from being linear (or quasi-linear) time computable is that the $t \times t$ circuit grid expands the number of gates---hence the number of clauses---quadratically. Claus-Peter Schnorr used a theorem in 1997-78 by Nicholas Pippenger and Mike Fischer that cuts the circuit sice to $O(t \log t)$, on the slight pain of making it have higher fan-out. That makes f computable in $O(p(n)\log p(n))$ time, where $p(n)$ is the running time of the verifier (or NTM) for A.

"SAT-like" Complete Problems

Some decision problems can be shown to be NP-hard or NP-complete by reductions that are "SAT-like." The first example uses the idea of a "mask" being a string of 0,1, and @ for "don't care". For instance, the mask string $s_0 \ = \ $@01@@0@@ forces the second bit to be 0, the third bit to be 1, and the sixth bit to be 0. A string like 00101001 "obeys" the mask, but 10011011 "violates" it in the third bit.

MASKS

Instance: A set of mask strings $s_1, \ \ldots, s_m$, all of the same length n.

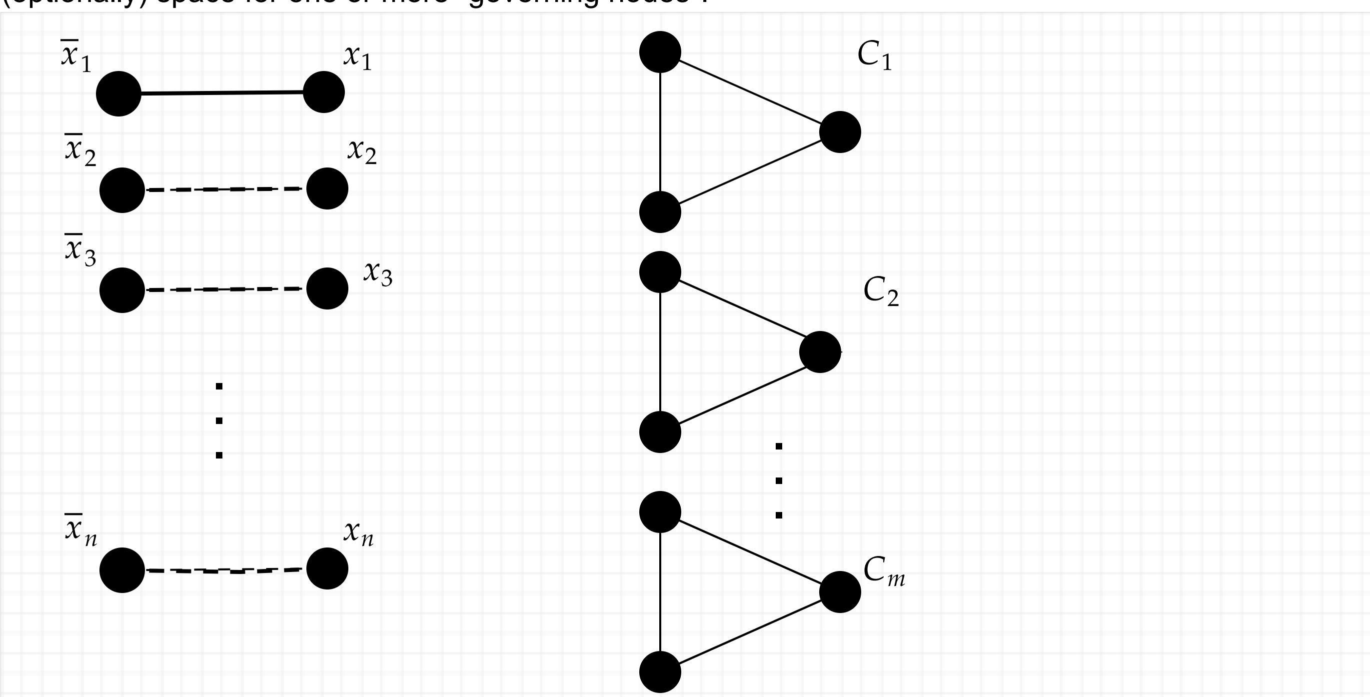
Question: Does there exist a string $a \ \in \ \{0, 1\}^n$ that violates **each** of the masks?

Then we get **3SAT** $\leq_m^p$ **MASKS** via a linear-time reduction f that converts each clause C_j to a mask s_j so that strings a that ***violate*** the mask are the same as assignments that ***satisfy*** C_j. For instance, if $C_j \ = \ (x_2 \ \vee \ \bar{x}_3 \ \vee \ x_6)$, then we get the mask $s_0 \ = \ $@01@@0@@ above. [This particular function f is invertible, so that we can readily get the clause from the mask, but it is important to keep in mind which direction the reduction is going in.]

Clearly the language of the **MASKS** problem is in NP, so it is NP-complete. We can also reduce **3TAUT** (whose instances are Boolean formulas ψ in *disjunctive normal form*, called **DNF**, having at most 3 literals per term) to the complementary problem of whether all strings x obey at least one mask. We can also make an NFA N_ψ that begins with ϵ-arcs to "lines" ℓ_j corresponding to each term T_j of ψ. Each line has n states that work to accept the strings x that obey the corresponding mask. Making N_ψ automatically accept all x of lengths other than n gives a reduction from **3TAUT** to the ALL_{NFA} problem, which finally explains why it is *hard*. (It is in fact not only co-NP hard under $\leq_m^p$ as this shows, but also NP-hard; it is in fact complete for the higher class PSPACE which we will get to next month.)

Reductions From 3SAT By Component Design - Part I

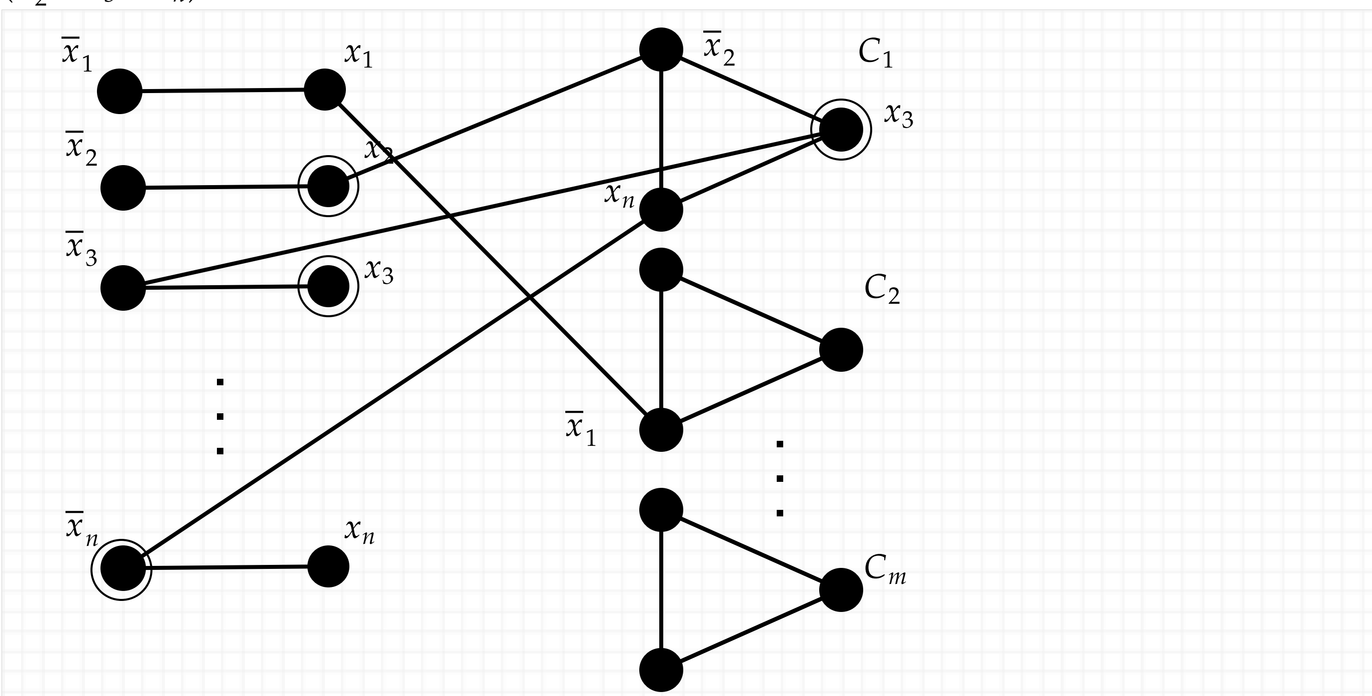
The **"Ladder and Gadgets"** framework for reduction from 3SAT: Given a 3CNF formula $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$, lay out n "rungs" of 2 nodes each and m "clause gadgets", plus (optionally) space for one or more "governing nodes":



Usually the rung nodes are connected, but not always---and sometimes an extra node or two are added to each rung.To show 3SAT $\leq_m^p$ IND SET, we need to map $f(\phi) = \langle G, k\rangle$ such that G has an independent set S of size (at least) k if and only if ϕ is satisfiable. Take $k = n + m$.

For this reduction, we make the "rungs" into actual edges between each x_i and its negation $\overline{x}_i$ and give each clause three nodes to make a triangle. Each clause node is labeled by a literal in the clause. Later we will include the clause index j, not just the variable index i, when identifying this *occurrence* of the literal in a clause to define V as a set, where $G = (V, E)$.

$(\overline{x}_2 \vee x_3 \vee x_n)$

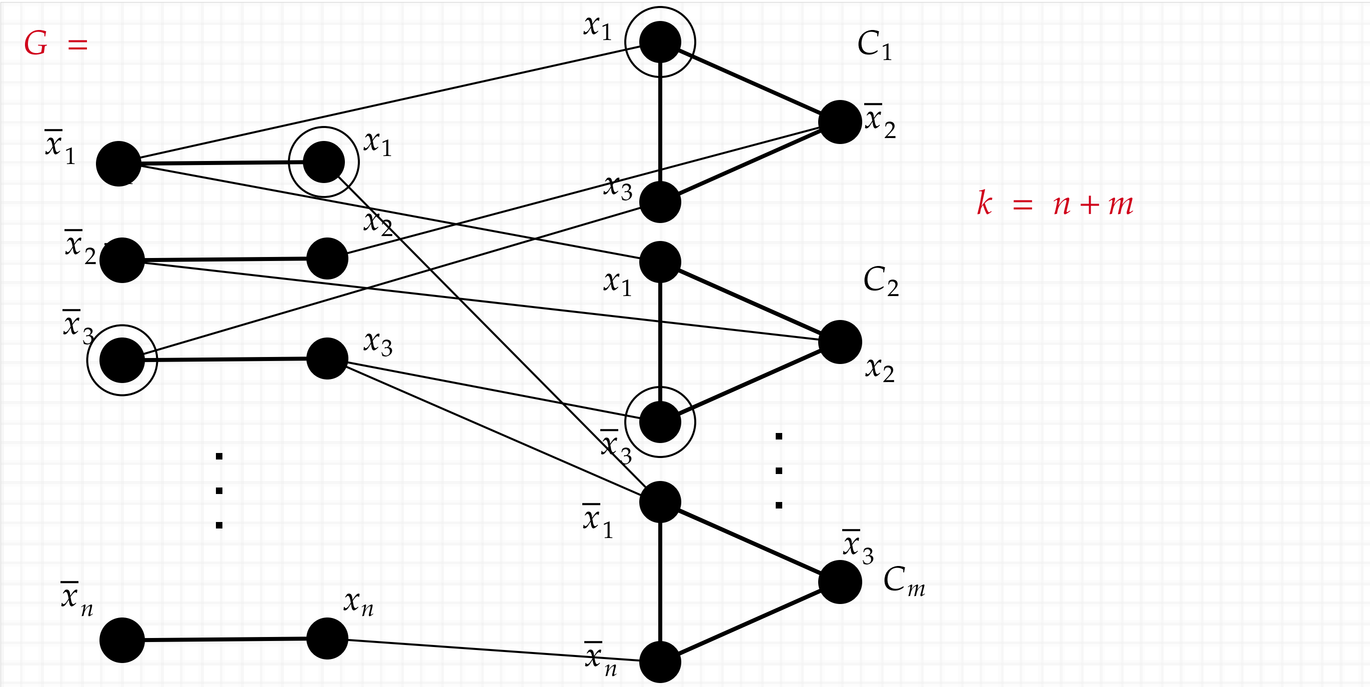


The immediate effect, even before we consider an example of a formula, is that the maximum possible k for an independent set S in the graph G is $n + m$. The most one can do is take one vertex from each rung and one from each triangle to make S. Note that the vertices chosen from each rung specify a truth assignment to the variables.

The final goal of the reduction is to add a third set of edges, which I call "crossing edges", to enforce that a set S of size $n + m$ is possible if and only if its corresponding assignment satisfies the formula. The basic idea, even before we consider a formula, is as follows.

- Suppose clause C_1 includes the positive literal x_1. Then we connect a crossing edge from x_1 in C_1 to the *opposite* literal $\overline{x}_1$ in the rung.
- Suppose clause C_2 includes the negated literal $\overline{x}_3$. Then we connect a crossing edge from $\overline{x}_3$ in C_2 to the opposite literal in the rung, which is just x_3.

$\phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4)$.



[Lecture ended here. Mon. Oct. 23, 2023, picked up with this example.]