GTI

Cook-Levin Theorem

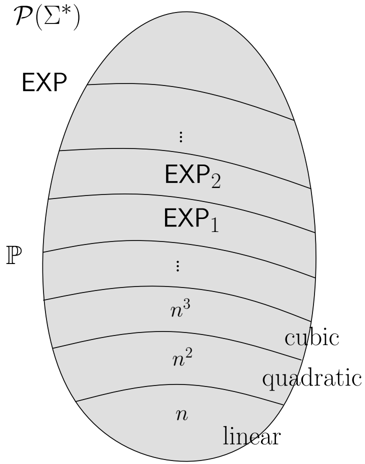
A. Ada, K. Sutner

Carnegie Mellon University

Spring 2018

1. Complexity
    - Cook-Levin

Our World


$\mathcal{P}(\Sigma^*)$
EXP
$\vdots$
EXP_2
EXP_1
$\mathbb{P}$
$\vdots$
n^3
cubic
n^2
quadratic
n
linear

“Intermediate” Problems

The picture on the last slide is quite useful, but there are a few annoying decision problems that don’t seem to quite fit.

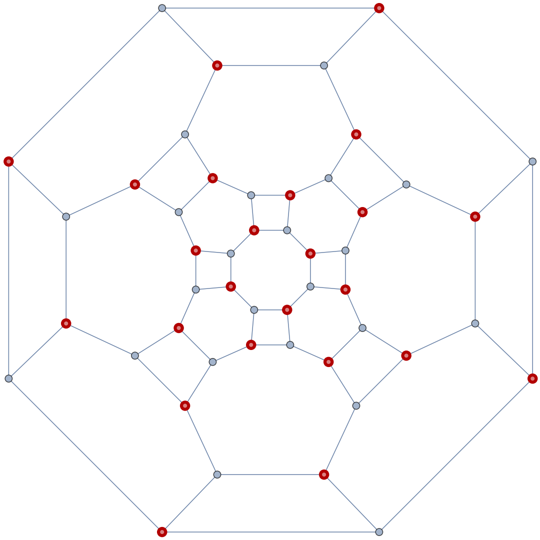
Problem:	**Vertex Cover (VC)**
Instance:	A ugraph G, a bound β.
Question:	Does G have a vertex cover of size β?

VC can easily be handled in exponential time (just check all subsets of V of size β), but no polynomial time algorithm is known to date (March 2018).

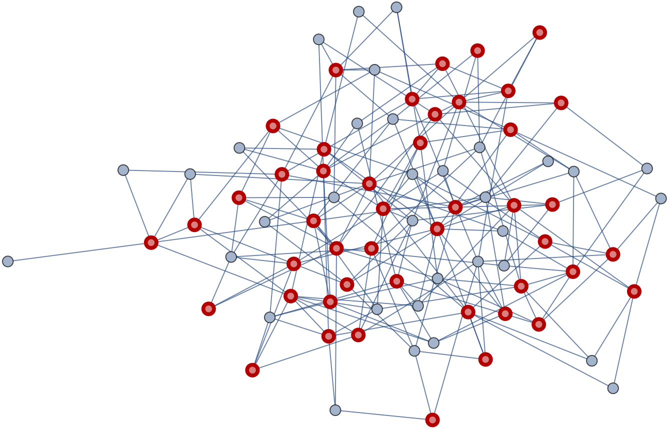
On the other hand, given a set $C \subseteq V$ it is trivial to verify in polynomial time that C is a vertex cover of size β.

This is very similar to the difference between finding a (short) proof and verifying its correctness.

A Symmetric VC



A Random VC



Recall: $\mathbb{NP}$

There are thousands of problems just like Vertex Cover: it is easy to check in polynomial time that a purported solution (a witness or certificate) is in fact correct, but finding one seems to require some kind of exponential search.

This behavior is enshrined in the complexity class $\mathbb{NP}$ for nondeterministic polynomial time.

To show that a problem L in in $\mathbb{NP}$ we need to find a polynomial time decidable predicate P such that

$$x \in L \iff \exists\, w\ (|w| \leq p(|x|) \wedge P(x,w))$$

where p is some polynomial.

Déjà Vu All Over Again

This should look eminently familiar by now:

L is semidecidable iff there is a decidable predicate P such that

$$x \in L \iff \exists\, w\, P(x, w)$$

The recursion theorists from the 1930s would have been quite comfortable with $\mathbb{NP}$ (except perhaps to question its very purpose).

Unfortunately, it seems that classical methods from computability theory have no impact on the $\mathbb{P}$ vs. $\mathbb{NP}$ question.

Reductions

One classical way to compare the difficulty of semidecidable sets is to use reductions: a problem B is harder than A if B can be used to solve A.

Turing Reduction $A \leq_T B$: B is available as an oracle in a decision algorithm for A.

Many-One Reduction $A \leq_m B$: there is a computable function f such that

$$x \in A \iff f(x) \in B$$

Lemma

Every semidecidable set is many-one reducible to the Halting Set.

Polynomial Time Reductions

Needless to say, these scale down into the world of complexity classes: we just add polynomial time bounds.

Cook Reduction polynomial time Turing reduction $A \leq_T^p B$: B is available as an oracle in a polynomial time decision algorithm for A.

Karp Reduction polynomial time (many-one) reduction $A \leq_m^p B$: there is a polynomial time computable function f such that

$$x \in A \iff f(x) \in B$$

Everything Works

Proposition

Both $\leq^p_T$ and $\leq^p_m$ are pre-orders (reflexive and transitive).

But Cook reductions are a bit too powerful for $\mathbb{NP}$: clearly $\overline{X} \leq^p_T X$, so we may have $A \leq^p_T B$ and $B \in \mathbb{NP}$, but $A \notin \mathbb{NP}$.

There is no such problem for Karp reductions:

Proposition

$A \leq^p_T B$ and $B \in \mathbb{NP}$ implies $A \in \mathbb{NP}$.

Of course, Karp reductions are sometimes a bit harder to find.

Hardness and Completeness

Definition

B is $\mathbb{NP}$-hard if for all A in $\mathbb{NP}$: $A \leq_m^p B$.

B is $\mathbb{NP}$-complete if B is in $\mathbb{NP}$ and is also $\mathbb{NP}$-hard.

The existence of an $\mathbb{NP}$-hard set is trivial: just take the disjoint union of all members of $\mathbb{NP}$.

$$A = \{\, x \# e \mid x \in L_e \,\}$$

where $(L_e)_e$ is any enumeration (not necessarily even effective) of all $\mathbb{NP}$ languages.

Alas, the existence of $\mathbb{NP}$-complete sets is far from obvious.

Pinpointing $\mathbb{NP}$

Proposition

If A in $\mathbb{NP}$-complete and A is in $\mathbb{P}$, then $\mathbb{P} = \mathbb{NP}$.

If indeed $\mathbb{P} \neq \mathbb{NP}$ as expected, then no $\mathbb{NP}$-complete problem can have a polynomial time algorithm.

This is the weak lower bound result that we are after: once a problem is known to by $\mathbb{NP}$-complete, one should be very reluctant to search for a polynomial time algorithm.

Constructing an $\mathbb{NP}$-complete Problem

How on earth do we get our hands on an $\mathbb{NP}$-complete problem?

> Given the analogy between semidecidable/decidable and $\mathbb{NP}/\mathbb{P}$ it is rather tempting to try to scale down the Halting Set (by adding polynomial time bounds).

OK as a battle-plan, but how exactly would this work?

We would want to use some universal machine $\mathcal{U}$ that can simulate all polynomial time verifiers, given by some suitable enumeration (N_e).

This requires the use of clocks (N_e runs in time $O(n^e + e)$) and some skulduggery to deal with witnesses (N_e magically guesses a witness, then verifies that there is no cheating). We will skip over the details.

Failure

A first shot would be to define

$$K = \{\, e \# x \mid x \text{ accepted by } N_e \,\}$$

It is easy to see that K is $\mathbb{NP}$-hard, but there is no reason why it should be in $\mathbb{NP}$; simulation of N_e is not a task that can be handled within a fixed polynomial time bound ($\mathcal{U}$ itself is not polynomial time).

So a simple head-on universality argument fails here.

This is more than slightly ironic: no bounds (classical recursion theory) really is much easier than polynomial bounds (complexity theory).

The slum is striking back.

An Ugly Set

We can fix the problem by padding the input so that we can compensate for the running time of N_e.

$$K = \{\, e \# x \# 1^t \mid x \text{ accepted by } N_e \text{ in } \leq t = |x|^e + e \text{ steps } \}$$

Proposition

K is in $\mathbb{NP}$.

Proof.

To see this note that the slowdown in the simulation of N_e by $\mathcal{U}$ is polynomial, say, the simulation takes $q(n^e + e)$ steps.

But then $\mathcal{U}$ can test, in time polynomial in $|e\#x\#1^t| = |x| + t + c$, whether N_e indeed accepts x.

□

Hardness

Proposition

K is $\mathbb{NP}$-hard.

Proof.

Consider $A = \mathcal{L}(N_e) \in \mathbb{NP}$ arbitrary. Then the function

$$x \mapsto e \# x \# 1^{|x|^e + |e|}$$

is polynomial time computable and shows that $A \leq_m^p K$.

□

Hence, K is indeed $\mathbb{NP}$-complete.

This Sucks

So we have the desired existence theorem.

Theorem

There is an $\mathbb{NP}$-complete language.

Alas, this result is perfectly useless when it comes to our list of $\mathbb{NP}$ problems: they bear no resemblance whatsoever to K.

We have a foothold in the world of $\mathbb{NP}$-completeness, but to show that one of these natural problems is $\mathbb{NP}$-complete we would have to find a reduction from K to, say, Vertex Cover.

Good luck on that.

- Complexity

2. Cook-Levin

A Better Mousetrap

We would like a “natural” $\mathbb{NP}$-complete decision problem: some concrete combinatorial problem (like Vertex Cover) that does not depend on obscure diagonalization tricks.

Where should one look for such a difficult problem?

Remember Hilbert’s dream: a general algorithm to solve the Entscheidungsproblem and answer all questions in mathematics?

Hilbert's Entscheidungsproblem

> *The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution of the Entscheidungsproblem is of fundamental importance for the theory of all fields, the theorems of which are at all capable of logical development from finitely many axioms.*
>
> *D. Hilbert, W. Ackermann*
> *Grundzüge der theoretischen Logik, 1928*

The Mother of All Problems

Instead of asking a single, concrete question (like: is there are vertex cover of some size), the Entscheidungsproblem asks all possible question in a certain domain (all theorems in graph theory).

Clearly this will be a major source of computational difficulty. In fact, in its full form, the Entscheidungsproblem is highly undecidable.

So, we need to scale it down a bit, so it winds up in $\mathbb{NP}$, but still maintains the flavor of "all possible questions."

The Key Problems

Problem:	**Satisfiability**
Instance:	A Boolean formula φ.
Question:	Is φ satisfiable?

Problem:	**Tautology**
Instance:	A Boolean formula φ.
Question:	Is φ a tautology?

SAT is clearly in $\mathbb{NP}$, TAUT in co-$\mathbb{NP}$.

Connections

Proposition

φ is satisfiable iff $\neg\varphi$ is not a tautology.

So a polynomial time algorithm for one would produce a polynomial time algorithm for the other (a Cook reduction).

But note that there is no obvious reason why Tautology should yield to the "guess-and-verify" approach that works so nicely for Satisfiability: what on earth should one guess?

Propositional Logic

Hilbert had first-order logic in mind when he talked about the Entscheidungsproblem: a system powerful enough to describe large fragments of mathematics (and currently the standard workhorse for proofs, the accepted standard for precision).

SAT scales all the way back to propositional logic, a much, much weaker system that is nowhere near expressive enough for most of mathematics. All we have is Boolean constants 0 and 1, Boolean variables and connectives $\neg$, $\wedge$ and $\vee$.

Still, a propositional formula with, say, 10,000 variables can express rather complicated assertions. Size matters.

And Circuits?

Boolean formulae are more or less the same as a circuits: the parse tree of a formula is essentially a circuit.

The inputs in a circuit correspond to the variables in a formula.

The output of a circuit corresponds to the value of the formula (given as input values for all the variables).

Recall DMP: 8 Queens

For example, it is not hard to concoct a formula

$$\Phi(x_{1,1}, x_{1,2}, \ldots, x_{8,8})$$

with 64 variables that expresses the assertion: we have an admissible placement of 8 queens on a chess board ($x_{i,j}$ is true iff there is a queen in position (i, j)).

Exercise

Build such a formula for an $n \times n$ chessboard and analyze its size.

Example: Vertex Cover

As another example, let us try to express the Vertex Cover problem as a SAT problem.

We have a ugraph $G = \langle V, E \rangle$ and a bound $\beta \leq n = |V|$.
Assume for simplicity that $V = [n]$ and $\beta < n$.

We use Boolean variables (a bitvector if you like) $\boldsymbol{x} = x_1, x_2, \ldots, x_n$ with the intended interpretation: x_i is true iff $i \in C$.

We construct a formula

$$\Phi(G, \beta) = \Phi_1 \wedge \Phi_2$$

where

- Φ_1 makes sure that C really is a vertex cover, and
- Φ_2 makes sure that C has size at most β.

The Formulae

The first part is easy:

$$\Phi_1(\boldsymbol{x}) = \bigwedge_{\{i,j\} \in E} x_i \vee x_j$$

For the second part we need a formula that says

$$\Phi_2(\boldsymbol{x}) = \text{ at most } \beta \text{ variables } x_i \text{ are true}$$

Unfortunately, the brute-force construction will produce a formula of exponential size:

$$\Phi_2(\boldsymbol{x}) = \neg \bigvee_{\boldsymbol{i}} (x_{i_1} \wedge x_{i_2} \wedge \ldots \wedge x_{i_{\beta+1}})$$

Faking Addition

To avoid exponential blow-up we use additional variables t_{ij}, $1 \leq i, j \leq n$, to count the number of true x variables. Then Φ_2 has the form

$$\bigwedge_{j>1} \neg t_{1j} \ \wedge \ \bigwedge_{i} \left(t_{i1} = \bigvee_{k \leq i} x_k \right) \wedge$$

$$\bigwedge_{i,j>1} \left(t_{ij} = (x_i \wedge t_{i-1,j-1} \wedge \neg t_{i-1,j}) \vee t_{i-1,j} \right) \wedge$$

$$\neg t_{n,\beta+1}$$

WTF?

Yup, this is plain dynamic programming.

The t-variables for $n = 6$ and $C = \{2, 4, 5\}$.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Make sure to trace the entries from the Boolean expression on the last slide.

In general, show by induction that

$$t_{ij} = 1 \iff \#(k \leq i \mid x_k = 1) \geq j$$

Done

Given the claim, $\neg t_{n,\beta+1}$ means that C contains at most β elements.

It is straightforward to see that Φ can be constructed from G and β in polynomial time: in essence, it suffices to note that Φ has size polynomial in n.

Lastly, by construction Φ is satisfiable iff G has a vertex cover of size β, so we have established a polynomial time reduction from VC to SAT.

This is not entirely frivolous, there are excellent SAT solvers, and it is tempting to apply them to other problems via reductions.

Cook-Levin Theorem

Back to work: we need to show that the last result is no coincidence, everything in $\mathbb{NP}$ reduces to SAT.

Theorem (Cook-Levin 1971/1973)

The Satisfiability Problem is $\mathbb{NP}$*-complete.*

Membership in $\mathbb{NP}$ is easy using the standard guess-and-verify approach.

But hardness takes work: we have to express an arbitrary $\mathbb{NP}$-problem as a Boolean formula.

Note that the following proof should be read just once. Then throw it away and reconstruct your own proof.

Proof Idea

Let A be an arbitrary set (of yes-instances) in $\mathbb{NP}$.

There is a deterministic polynomial time Turing machine M such that M accepts (x, w) iff $x \in A$ where w is some witness of length polynomial in $n = |x|$.

The idea is to construct a (rather large) Boolean formula Φ_x such that

$$\Phi_x \text{ is satisfiable } \iff M \text{ accepts } (x, w) \text{ for some } w.$$

While the formula is fairly long, it has a clear structure and it can easily be constructed in time polynomial in n.

It has lots of variables that express information about states, head position and tape inscriptions: the satisfying truth assignment translates directly into an accepting computation of M.

Coding Time

First off, let $N = p(n)$ be the running time of the machine.

If we have a list of Boolean variables

$$X_0, X_1, \ldots, X_N$$

and a truth assignment σ we can think of $\sigma(X_t)$ as the value of variable X at time t.

We can use logic to pin down the value of X_{t+1} in terms of X_t (and other variables).

$$\bigwedge_{t<N} X_{t+1} \iff \varphi(X_t, \ldots)$$

Coding Numbers

If we need to code a number r in a certain range, say $1 \leq r \leq m$, we can simply use variables

$$X(1), X(2), \ldots, X(m)$$

plus a stipulation that exactly one of them is true under σ:

$$\mathsf{EO}_m\big(X(1), X(2), \ldots, X(m)\big)$$

Here

$$\mathsf{EO}_k(x_1, \ldots, x_k) = (x_1 \vee x_2 \ldots \vee x_k) \wedge \bigwedge_{1 \leq i < j \leq k} \neg(x_i \wedge x_j).$$

Note that the size of EO_k is $O(k^2)$.

Coding Hardware

Combining these two ideas we can set up polynomially many Boolean variables

states	$S_t(p)$, $0 \leq t \leq N$, $1 \leq p \leq m$,
head position	$H_t(i)$, $0 \leq t, i \leq N$,
tape inscription	$C_t(i)$, $0 \leq t, i \leq N$.

that express, for each time $0 \leq t \leq N$, which state the machine is in, where the head is, and what's on the tape.

For simplicity, we assume here that the only tape symbols are 0 and 1, it is not hard to deal with the general situation.

Coding Moves

We then have to express the constraint that the variables change from time t to time $t+1$ only in accordance with the transition function of the Turing machine.

For example

$$\bigwedge_{t<N} H_t(i) \Rightarrow \mathsf{EO}_3\big(H_{t+1}(i{-}1), H_{t+1}(i), H_{t+1}(i{+}1)\big)$$

$$\bigwedge_{t<N} S_t(p) \land H_t(i) \land \neg C_t(i) \Rightarrow S_{t+1}(q_0)$$

$$\bigwedge_{t<N} S_t(p) \land H_t(i) \land C_t(i) \Rightarrow S_{t+1}(q_1)$$

And so on and on.

Start and Stop

Initially the input is on the first part of the tape

$$H_0(0) \wedge S_0(q_0) \wedge C_0(1) = x_1 \wedge C_0(2) = x_2 \wedge \ldots \wedge C_0(n) = x_n$$

and at the end we accept:

$$S_N(q_Y)$$

Note that $C_0(n+1), \ldots, C_0(N)$ is not fixed and can be set arbitrarily by σ.

It is not too hard to see that the whole formula Φ_x in the end has size polynomial in n.

And It Works

Now suppose Φ_x is satisfied by truth assignment σ.

Then M accepts the original tape inscription (x, w) where x is the real input and w is the bits chosen freely by σ.

Since Φ_x forces the values of all the variables to correspond to a computation of M on input (x, w) we have the desired witness and x must be a Yes-instance.

Conversely, every witness plus corresponding accepting computation can be translated into a satisfying truth assignment σ.

That's it. $\square$

Improvements

The hardness of Satisfiability holds up even when the formulae in question are rather restricted.

Definition

A literal is a variable or negated variable. A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals:
$\Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_n$ where $\Phi_i = z_{i,1} \vee z_{i,2} \vee \ldots \vee z_{i,k(i)}$, $z_{i,j}$ a literal.

It is in k-CNF if $k(i) = k$ for all i.

In particular 3-CNF is very popular. But note that this is the limit:

Exercise

Show that 2-CNF is solvable polynomial time.

Refinement

Theorem

The Satisfiability Problem is $\mathbb{NP}$-complete for formulae in 3-CNF.

Proof.

Bring in the shovel brigade: check the Cook-Levin proof carefully to show that the formula produced there can easily (meaning: in polynomial time) be converted into 3-CNF.

□

Beachhead

The last result can be used to establish the $\mathbb{NP}$-hardness of many problems such as Vertex Cover.

It now suffices to find a polynomial time computable function

$$f : \text{3-CNF} \longrightarrow \mathrm{Graphs} \times \mathrm{Integers}$$

such that φ in 3-CNF is satisfiable iff for $f(\varphi) = (G, \beta)$ the graph G has a vertex cover of size β.

This is much, much easier than having to deal with general formulae.

Spreading Completeness

Satisfiability is a tremendously important practical problem, but if this were the only relevant $\mathbb{NP}$-complete problem the whole notion would still be somewhat academic.

But as Dick Karp realized after reading Cook's paper, there are dozens (actually: thousands) of combinatorial problems that all turn out to be $\mathbb{NP}$-complete. So none of them will admit a polynomial time solution unless $\mathbb{P} = \mathbb{NP}$.

The proof method is interesting: some problems are proven hard by direct reduction from SAT, then these are used to show other problems are hard, and so on ... By transitivity one could, in principle, produce a direct reduction from SAT, but in reality these direct reductions are often very hard to find.

Theorem

Vertex Cover is $\mathbb{NP}$*-complete.*

Proof.

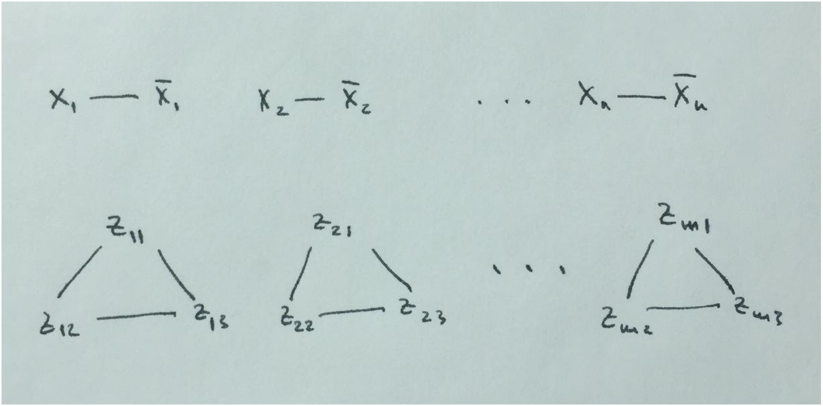
Suppose we have a 3-CNF formula $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_m$ where $\Phi_i = z_{i,1} \vee z_{i,2} \vee z_{i,3}$.

The Boolean variables are $x_1, \ldots, x_n$.

We start with a graph G' on $2n + 3m$ vertices.

- Vertices: x_i, $\overline{x}_i$ for $i = 1, \ldots, n$ and $u_{i,1}$, $u_{i,2}$, $u_{i,3}$ for $i = 1, \ldots, m$.
- Edges: one edge between x_i and $\overline{x}_i$, and three edges that turn $u_{i,1}$, $u_{i,2}$, $u_{i,3}$ into a triangle.

Picture



Proof, cont'd.

It is easy to see that every vertex cover of G' must have at least $n + 2m$ vertices (one for each x-edge, and two for each triangle).

And such covers exist, lots of them: you can pick at random one of x_i or $\overline{x}_i$, and exactly two of $u_{i,1}$, $u_{i,2}$, $u_{i,3}$.

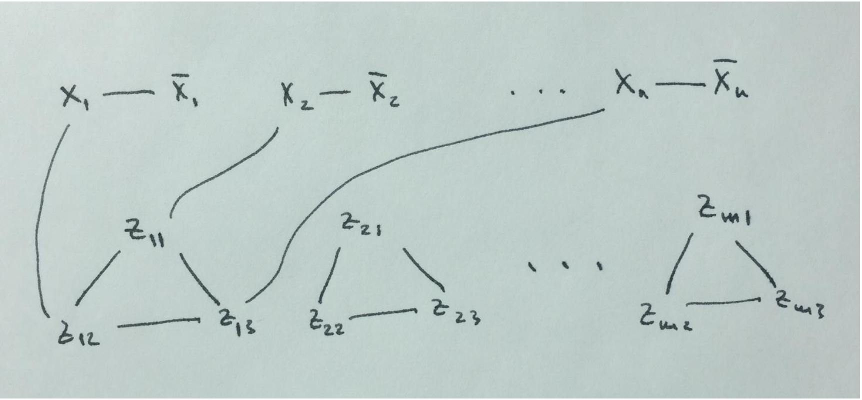
So far we have only used n and m, but not the formula itself.

Let G be the graph obtained by adding $3m$ more edges to G':

connect $u_{i,j}$ to x_s if $z_{i,j} = x_s$ and connect $u_{i,j}$ to $\overline{x}_s$ if $z_{i,j} = \neg x_s$.

Lastly, set the bound to $\beta = n + 2m$.

Picture



Proof, cont'd.

Claim

G has a cover of size $\beta = n + 2m$ iff the formula is satisfiable.

To see this, note that any cover C defines an assignment σ:

$$\sigma(x_i) = \begin{cases} 1 & \text{if } x_i \in C, \\ 0 & \text{otherwise.} \end{cases}$$

Then σ satisfies the formula.

Conversely, every satisfying assignment translates into a cover.

□

Important Points

For this construction to work we need two crucial ingredients:

- The graph G and the bound β can be computed from Φ in polynomial time.
- G has a vertex cover of size β if, and only if, Φ is satisfiable.

Many other completeness proofs look very similar: it is trivial to see that the problem is in $\mathbb{NP}$, and it requires work (sometimes a lot of it) to produce hardness.

Cheap Shots

Problem:	**Independent Set (IS)**
Instance:	A ugraph G, a bound β.
Question:	Does G have an independent set of size β?

Problem:	**Clique**
Instance:	A ugraph G, a bound β.
Question:	Does G have a clique of size β?

These look rather similar to VC. Is there an algorithmic connection?

All The Same

Lemma

VC, IS and Clique are all Karp reducible to each other.

Proof. Let $G = \langle V, E \rangle$ be a graph, G^c its complement, and $C \subseteq V$. The following are equivalent:

- C is a vertex cover of G.
- $V - C$ is an independent set of G.
- $V - C$ is a clique of G^c.

Hence $f(G, \beta) = (G^c, n - \beta)$ is a reduction from VC to Clique. Rest similar.