Harvard CS 121 and CSCI E-121

Lecture 23: More NP-completeness

Harry Lewis

November 21, 2013

More NP-complete problems

From now on we prove NP-completeness using:

Lemma: If we have the following

- L is in NP
- $L_0 \leq_{\mathsf{P}} L$ for some NP-complete L_0

Then L is NP-complete.

Proof:

3-SAT

Def: A Boolean formula is in 3-CNF if it is of the form:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each clause C_i is a disjunction ("or") of 3 literals:

$$C_i = (C_{i1} \vee C_{i2} \vee C_{i3})$$

where each literal C_{ij} is either

- a variable x, or
- the negation of a variable, $\neg x$.

e.g. $(x \vee y \vee z) \wedge (\neg x \vee \neg u \vee w) \wedge (u \vee u \vee u)$

3-SAT is the set of satisfiable 3-CNF formulas.

A Note on Conjunctive Normal Form

- Every boolean formula can be put into CNF by using DeMorgan's Laws
- e.g. $(a \wedge b) \vee (c \wedge d) \equiv (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
- So why isn't it obvious that CNF-SAT is NP-complete??

3-SAT is NP-complete

Proof: Show that SAT $\leq_P$ 3-SAT.

1. Given an arbitrary Boolean formula, e.g.

$$F = (\neg((x \vee \neg y) \wedge (z \vee w)) \vee \neg x).$$
$$\quad 1 \quad 2\ 3 \quad 4 \quad 5 \quad 6\ 7$$

2. Number the operators.

3. Select a new variable a_i for each operator.
   The variable a_i is supposed to mean "the subformula rooted at operator i is true."

4. Write a formula stating the relation between each subformula and its children subformulas.

Reduction of SAT to 3-SAT, continued

For example, where

$$F = (\neg((x \vee \neg y) \wedge (z \vee w)) \vee \neg x),$$
$$\quad 1 \quad 2\ 3 \quad 4 \quad 5 \quad 6\ 7$$

$$F_1 = \left(\begin{array}{lcl} & (a_3 \equiv \neg y) & \wedge \quad (a_7 \equiv \neg x) \\ \wedge & (a_2 \equiv x \vee a_3) & \wedge \quad (a_1 \equiv \neg a_4) \\ \wedge & (a_5 \equiv z \vee w) & \wedge \quad (a_6 \equiv a_1 \vee a_7) \\ \wedge & (a_4 \equiv a_2 \wedge a_5) & \end{array} \right)$$

5. Let k be the number of the main operator/subformula of F. (Note: $k = 6$ in the example)

Write F_1 in 3-CNF to obtain F_2

- **Fact:** Every function $f : \{0,1\}^k \to \{0,1\}$ can be written as a k-CNF and as a k-DNF (OR of ANDs).
  [albeit with possibly 2^k clauses]

- **Proof:**

Output of the reduction: $a_k \wedge F_2$.

Q: Does this prove that every Boolean formula can be converted to 3-CNF?

In contrast, 2-SAT $\in$ P

Method (resolution):

1. If x and $\neg x$ are both clauses, then not satisfiable

   e.g. $(x) \wedge (z \vee y) \wedge (\neg x)$

2. If $(x \vee y) \wedge (\neg y \vee z)$ are both clauses, add clause $(x \vee z)$ (which is implied).

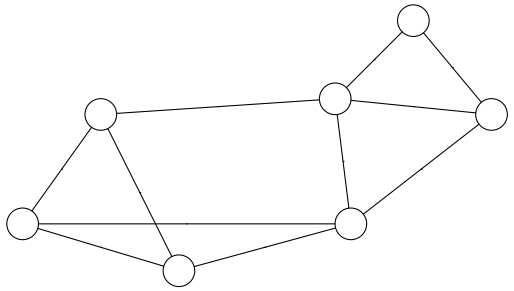
3. Repeat. If no contradiction emerges $\Rightarrow$ satisfiable.

$O(n^2)$ repetitions of step 2 since only 2 literals/clause.

Proof of correctness: omitted

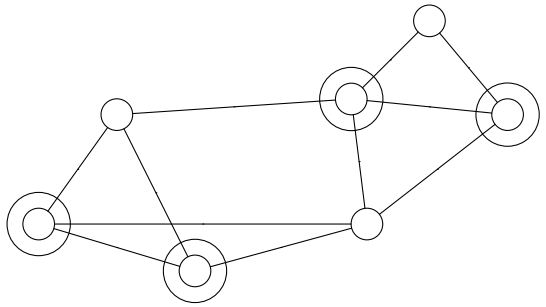
VERTEX COVER (VC)

- Instance:
  - a graph, e.g.



  - a number k (e.g. 4)

- Question: Is there a set of k vertices that "cover" the graph, i.e., include at least one endpoint of every edge?



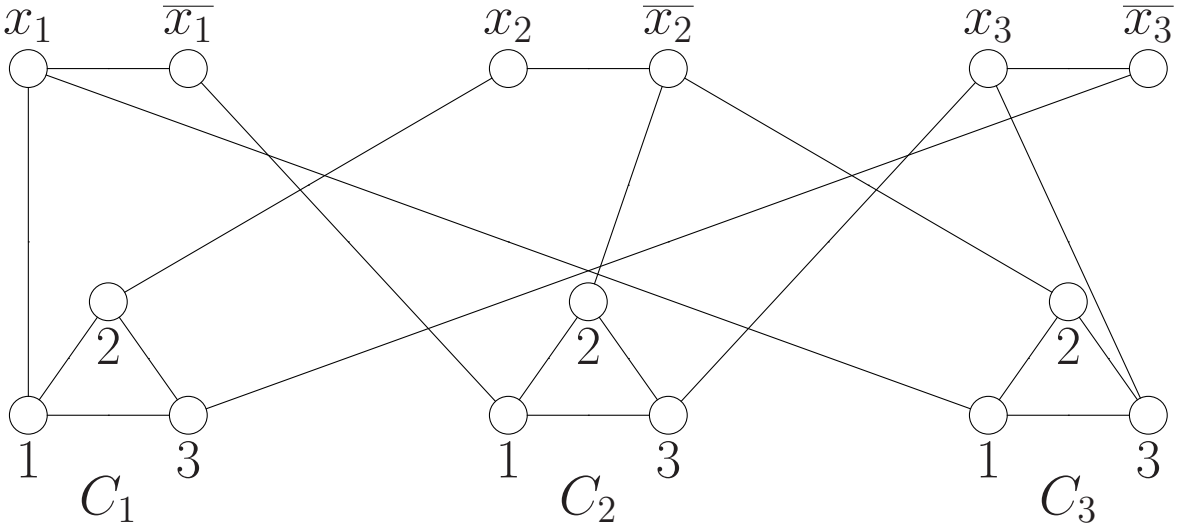
VC is NP-complete

- VC is in NP:
- 3-SAT $\leq_P$ VC:
  - Let F be a 3-CNF formula with clauses $C_1 \ldots, C_m$, variables $x_1, \ldots, x_n$.
  - We construct a graph G_F and a number N_F such that:

    G_F has a size N_F vertex cover iff F is satisfiable

Construction of G_F and N_F from F

E.g. $F = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$



- G_F = one dumbbell for each variable, one triangle for each clause, and corner j of triangle i is connected to the vertex representing the jth literal in C_i.

- $N_F = 2m + n = 2$ (# clauses) $+$ (# variables).
  $\Rightarrow$ 1 vertex from each dumbbell and 2 from each triangle.

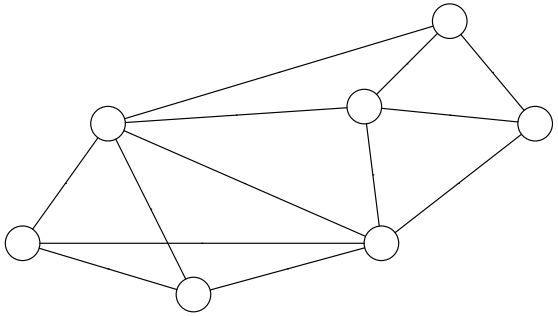
Correctness of the Reduction

- If F is satisfiable, then there is an N_F cover:

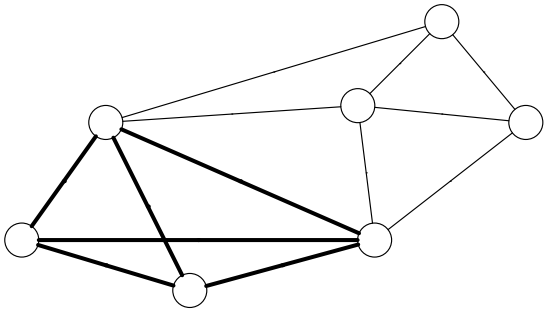
- If there is an N_F cover, then F is satisfiable:

CLIQUE

- Instance:
  - a graph, e.g.



  - a number k (e.g. 4)
- Question: Is there a clique of size k, i.e., a set of k vertices such that there is an edge between each pair?



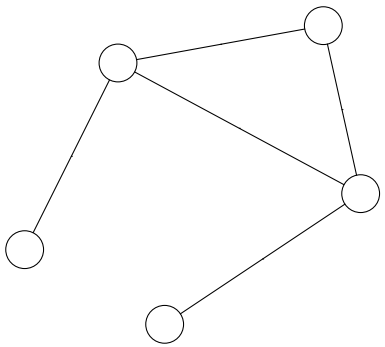
- Easy to see that CLIQUE $\in$ NP.

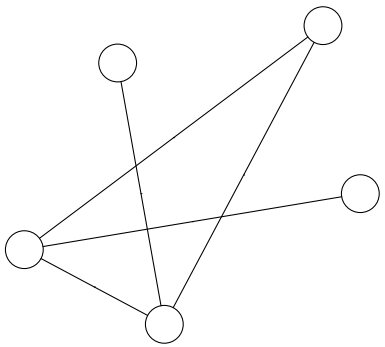
VC $\leq_P$ CLIQUE

If G is any graph, let G^c be the graph with the same vertices such that:

there is an edge between x and y in G^c
iff
there is no edge between x and y in G

e.g.

$G =$ 

$G^c =$ 

VC $\leq_P$ CLIQUE, continued

Let (G, k) be an instance of VC.

Claim: G has a k-cover iff G^c has a $|G| - k$ clique,
where $|G|$ is the number of vertices in G.

(So the mapping $(G, k) \mapsto (G^c, |G| - k)$
is a reduction of VC to CLIQUE.)

Proof:

INTEGER LINEAR PROGRAMMING

An integer linear program is

- A set of variables $x_1, \ldots, x_n$ which must take integer values.
- A set of linear inequalities:
  $a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n \leq c_i \qquad [i = 1, \ldots, m]$

e.g. $x_1 - 2x_2 + x_4 \leq 7$

$x_1 \geq 0 \qquad [-x_1 \leq 0]$

$x_4 + x_1 \leq 3$

ILP $=$ the set of integer linear programs for which there are values for the variables that simultaneously satisfy all the inequalities.

ILP is NP-complete

INTEGER LINEAR PROGRAMMING $\in$ NP. (Not obvious! Need a little math to prove it. Proof omitted.)

INTEGER LINEAR PROGRAMMING is NP-hard: by reduction from 3-SAT ($3\text{-SAT} \leq_P \text{ILP}$). Given 3-CNF Formula F, construct following ILP P as follows:

Note: LINEAR PROGRAMMING where the variables can take *real* values is known to be in P.

More NP-complete/NP-hard Problems

- HAMILTONIAN CIRCUIT (and hence TSP) (see Sipser for related problems)
- SCHEDULING
- CIRCUIT MINIMIZATION
- SHORT PROOF
- NASH EQUILIBRIUM WITH MAXIMUM PAYOFF
- PROTEIN FOLDING
- ⋮
- See Garey & Johnson for hundreds more.