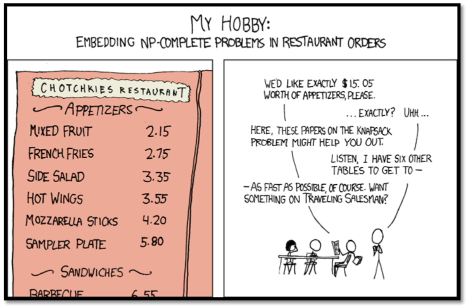
CSE 421: Algorithms

Winter 2014

Lecture 23: P, NP, and reductions

Reading:
Sections 8.3-8.7



polynomial time

Define P (polynomial-time) to be

- the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.

beyond P?

- There are many other natural, practical problems for which we don't know any polynomial-time algorithms
- For example: decisionTSP
  - Given a weighted graph G and an integer k, does there exist a tour that visits all vertices in G having total weight at most k?

satisfiability

- Boolean variables $x_1,\ldots,x_n$
  - taking values in {0,1}. 0=false, 1=true
- Literals
  - x_i or $\neg x_i$ for i=1,…,n
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses
- k-CNF formula
  - All clauses have exactly k variables

satisfiability

- CNF formula example
  $(x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3)$
- If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable
  - the one above is, the following isn't
  - $x_1 \land (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \neg x_3$
- 3-SAT: Given a CNF formula F with 3 variables per clause, is it satisfiable?

common property of these problems

- There is a special piece of information, a short certificate or proof, that allows you to **efficiently verify** (in polynomial-time) that the YES answer is correct. This certificate might be very hard to find
- e.g.
  - **DecisionTSP**:
  - **Independent-Set**, **Clique**:
  - **3-SAT**:

The complexity class NP

NP consists of all decision problems where

- You can **verify** the YES answers efficiently (in polynomial time) given a short (polynomial-size) certificate

and

- No certificate can fool your polynomial time verifier into saying YES for a NO instance

more precise definition of NP

- A decision problem is in NP iff there is a polynomial time procedure **verify**(.,.), and an integer k such that
  - for every input x to the problem that is a YES instance there is a certificate t with $|t| \leq |x|^k$ such that **verify**(x,t) = YES

  and
  - for every input x to the problem that is a NO instance there does not exist a certificate t with $|t| \leq |x|^k$ such that **verify**(x,t) = YES

CLIQUE is in NP

procedure **verify**(**x**,**t**)
  if **x** is a well-formed representation of
    a graph **G** = (**V**, **E**) and an integer **k**,
  and
    t is a well-formed representation of a vertex subset **U** of **V** of size **k**,
  and
    U is a clique in **G**,
  then output "**YES**"
  else output "**I'm unconvinced**"

is it correct?

keys to showing a problem is in NP

- What's the output? (must be YES/NO)
- What must the input look like?
- Which inputs need a YES answer?
  - Call such inputs YES inputs/YES instances
- For every given YES input, is there a certificate that would help?
  - OK if some inputs need no certificate
- For any given NO input, is there a fake certificate that would trick you?

solving NP problems without hints

The only **obvious algorithm** for most of these problems is **brute force**:

- try all possible certificates and check each one to see if it works.
- *Exponential* time:
  - 2^n truth assignments for **n** variables
  - **n!** possible TSP tours of **n** vertices
  - $\binom{n}{k}$ possible **k** element subsets of **n** vertices
  - etc.

what we know

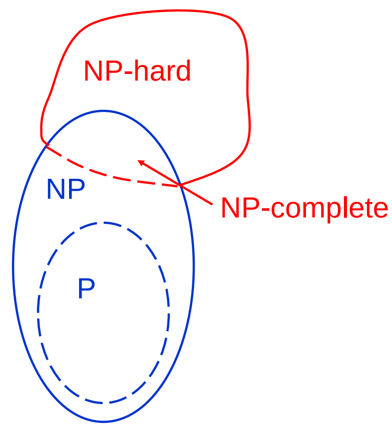
- Nobody knows if all problems in **NP** can be done in polynomial time, i.e. does **P = NP ?**
  - one of the most important open questions in all of science.
  - huge practical implications
- Every problem in **P** is in **NP**
- Every problem in **NP** can be solved in exponential time

solving NP problems in exponential time

NP-hardness & NP-completeness

- Alternative approach to proving problems not in **P**
  - show that they are at least as hard as any problem in **NP**
- Rough definition:
  - A problem is **NP-hard** iff it is at least as hard as any problem in **NP**
  - A problem is **NP-complete** iff it is both
    - **NP**-hard
    - in **NP**

P and NP



NP-hardness & NP-completeness

- Definition: A problem **B** is NP-hard iff every problem $A \in NP$ satisfies $A \leq_P B$
- Definition: A problem **B** is NP-complete iff **A** is NP-hard and $A \in NP$
- Even though we seem to have lots of hard problems in NP it is not obvious that such super-hard problems even exist!

Cook-Levin Theorem

- Theorem (Cook 1971, Levin 1973):

  3-SAT is NP-complete.

- Recall
  - CNF formula

    $(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$
  - If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable
  - 3-SAT: Given a 3-CNF formula F, is it satisfiable?

implications of the Cook-Levin theorem?

- There is at least one interesting super-hard problem in NP
- Is that such a big deal?
- Yes, a jumping off point.
  - There are lots of other problems that can be solved if we had a polynomial-time algorithm for 3-SAT
  - Many of these problems are exactly as hard as 3-SAT

A useful property of polynomial-time reductions

- Theorem: If $A \leq_P B$ and $B \leq_P C$ then $A \leq_P C$
- Proof idea: (Using $\leq_P^1$)
  - Compose the reduction **f** from **A** to **B** with the reduction **g** from **B** to **C** to get a new reduction $h(x)=g(f(x))$ from **A** to **C**.
  - The general case is similar and uses the fact that the composition of two polynomials is also a polynomial

A useful property of polynomial-time reductions

- **Theorem: If $A \le_P B$ and $B \le_P C$ then $A \le_P C$**
- **Proof idea:**

Cook-Levin theorem & implications

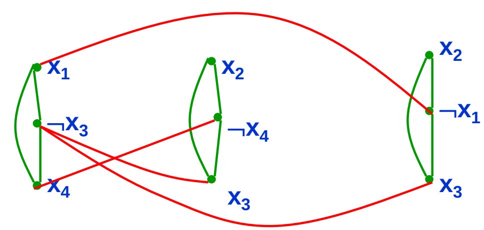
- **Theorem (Cook 1971, Levin 1973):**
  3-SAT is NP-complete (for proof see CSE 431)
- **Corollary: B is NP-hard ⇔ 3-SAT $\le_P$ B**
  (or A $\le_P$ B for any NP-complete problem A)
- **Proof:**
  - If B is NP-hard then every problem in NP polynomial-time reduces to B, in particular 3-SAT does since it is in NP
  - For any problem A in NP, A $\le_P$ 3-SAT and so if 3-SAT $\le_P$ B we have A $\le_P$ B.
    therefore B is NP-hard if 3-SAT $\le_P$ B

3-SAT $\le_P$ Independent-Set

- **A Tricky Reduction:**
  - mapping CNF formula F to a pair <G,k>
  - Let m be the number of clauses of F
  - Create a vertex in G for each literal in F
  - Join two vertices u, v in G by an edge iff
    u and v correspond to literals in the same clause of F, (green edges) or
    u and v correspond to literals x and ¬x (or vice versa) for some variable x. (red edges)
  - Set k=m
  - Clearly polynomial-time

3-SAT $\le_P$ Independent-Set

F: $(x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3)$



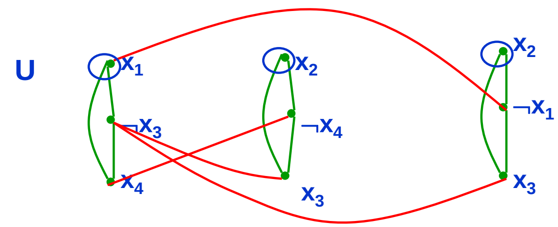
3-SAT $\leq_P$ Independent-Set

- **Correctness:**
  - **If F is satisfiable then there is some assignment that satisfies at least one literal in each clause.**
  - **Consider the set U in G corresponding to the first satisfied literal in each clause.**
    - $|U|=m$
    - Since U has only one vertex per clause, no two vertices in U are joined by green edges
    - Since a truth assignment never satisfies both x and $\neg x$, U doesn't contain vertices labeled both x and $\neg x$ and so no vertices in U are joined by red edges
    - Therefore G has an independent set, U, of size at least m
  - **Therefore (G,m) is a YES for independent set.**

3-SAT $\leq_P$ Independent-Set

1 0 1 1 0 1 1 0 1

F: $(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$



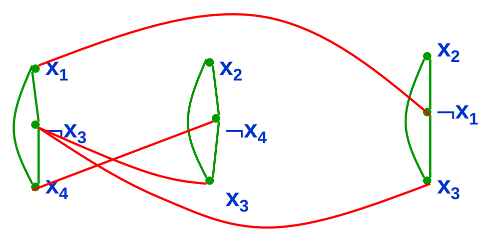
Given assignment $x_1=x_2=x_3=x_4=1$,
U is as circled

3-SAT $\leq_P$ Independent-Set

- **Correctness continued:**
  - **If (G,m) is a YES for Independent-Set then there is a set U of m vertices in G containing no edge.**
    - Therefore U has precisely one vertex per clause because of the green edges in G.
    - Because of the red edges in G, U does not contain vertices labeled both x and $\neg x$
    - Build a truth assignment A that makes all literals labeling vertices in U true and for any variable not labeling a vertex in U, assigns its truth value arbitrarily.
    - By construction, A satisfies F
  - **Therefore F is a YES for 3-SAT.**

3-SAT $\leq_P$ Independent-Set

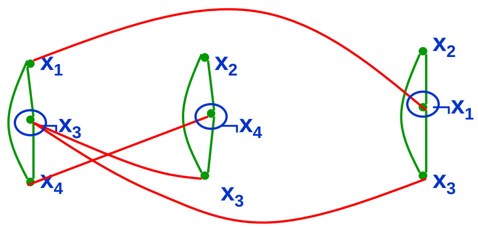
F: $(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_3)$



3-SAT $\leq_P$ Independent-Set

0 1 0 ? 1 0 ? 1 0

F: $(x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3)$



Given **U**, satisfying assignment is $x_1=x_3=x_4=0$, $x_2=0$ or 1

Independent-Set is NP-complete

- We just showed that Independent-Set is NP-hard and we already knew Independent-Set is in NP.
- Corollary: Clique is NP-complete
  - We showed already that Independent-Set $\leq_P$ Clique and Clique is in NP.