Harvard CS 121 and CSCI E-207

Lecture 22: More NP-Completeness and the Cook-Levin Theorem

Harry Lewis

November 30, 2010

Last time, we showed how to reduce SAT to other problems:

$$\text{SAT} \le_P \text{3-SAT} \le_P \textsc{Vertex Cover} \le_P \text{CLIQUE}$$

Today,

1. One more problem to which SAT can be reduced
2. Show that every NP problem is reducible to SAT
3. A view of the whole space of problems as we now understand it
4. A few words about space complexity

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 $(3\text{-SAT} \leq_{\mathsf{P}} \text{ILP})$. 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
- $\vdots$
- See Garey & Johnson for hundreds more.

Cook-Levin Theorem: SAT is NP-complete

Proof:

- Already know SAT $\in$ NP, so only need to show SAT is NP-hard.
- Let L be any language in NP. Let M be a NTM that decides L in time n^k.
  We define a polynomial-time reduction

  $$f_L : \text{inputs} \mapsto \text{formulas}$$

  such that for every w,

  $$M \text{ accepts input } w \text{ iff } f_L(w) \text{ is satisfiable}$$

Reduction via "computation histories"

Proof Idea: satisfying assignments of $f_L(w) \leftrightarrow$ accepting computations of M on w

<u>Describe computations of M by boolean variables:</u>

- If $n = |w|$, then any computation of M on w has at most n^k configurations.
- Each configuration is an element of C^{n^k}, where $C = Q \cup \Gamma \cup \{\#\}$ (mark left and right ends with $\{\#\}$).

$\rightsquigarrow$ computation depicted by $n^k \times n^k$ "tableau" of members of C.

- Represent contents of cell (i, j) by $|C|$ boolean variables $\{x_{i,j,s} : s \in C\}$, where $x_{i,j,s} = 1$ means "cell (i, j) contains s".
- $0 \leq i, j < n^k$, so $|C| \cdot n^{2k}$ boolean variables in all

Subformulas that verify the computation

Express conditions for an accepting computation on w by boolean formulas:

- $\phi_{\text{cell}} =$
  "for each (i, j), there is exactly one $s \in C$ such that $x_{i,j,s} = 1$".

- $\phi_{\text{start}} =$ "first row equals start configuration on w"

- $\phi_{\text{accept}} =$ "last row is an accept configuration on w"

- $\phi_{\text{move}} =$ "every 2×3 window is consistent with the transition function of M"

Completing the proof

Claim: Each of above can be expressed by a formula of size of size $O((n^k)^2) = O(n^{2k})$, and can be constructed in polynomial time from w.

Claim: M has an accepting computation on w if and only if $f_L(w) = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$ has a satisfying assignment.

Thus $w \mapsto f_L(w)$ is a polynomial-time reduction from L to SAT.

Since above holds for every $L \in$ NP, SAT is NP-hard, as desired. ■

Clay Mathematics Institute

Dedicated to increasing and disseminating mathematical knowledge

HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

P vs NP Problem

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

- The Millennium Problems
- Official Problem Description — Stephen Cook
- Lecture by Vijaya Ramachandran at University of Texas (video)
- Minesweeper

A Proof That P Is Not Equal To NP?

AUGUST 8, 2010

by rjlipton tags: P=NP, Proo

A serious proof that claims to have resolved the P=NP question.

Vinay Deolalikar is a Principal Research Scientist at HP Labs who has done important research in various areas of networks. He also has worked on complexity theory, including previous work on **infinite** versions of the P=NP question. He has just claimed that he has a proof that P is not equal to NP. That's right: $P \neq NP$. No infinite version. The real deal.

Today I will talk about his paper. So far I have only had a chance to glance at the paper; I will look at it more carefully in the future. I do not know what to think right now, but I am certainly hopeful.

The Paper

co-NP

Recall that co-NP $= \{\overline{L} : L \in$ NP$\}$.

Some co-NP-complete problems:

- Complement of any NP-complete problem.
- TAUTOLOGY = $\{\varphi : \forall a\ \varphi(a) = 1\}$ (even for 3-DNF formulas φ).

Believed that NP $\neq$ co-NP, P $\neq$ NP $\cap$ co-NP.

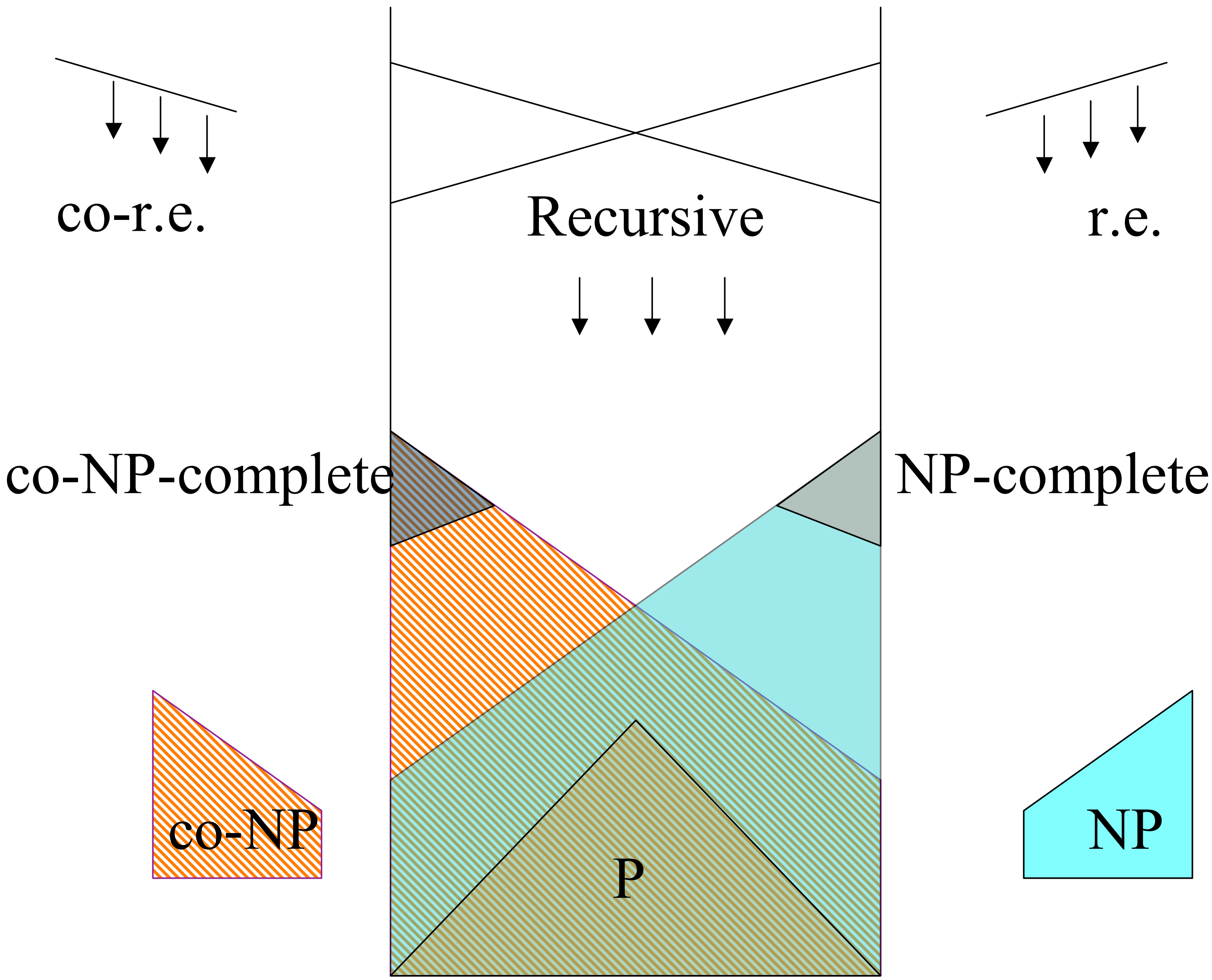
Between P and NP-complete

Theorem: If P $\neq$ NP, then there are NP languages that are neither in P nor NP-complete.

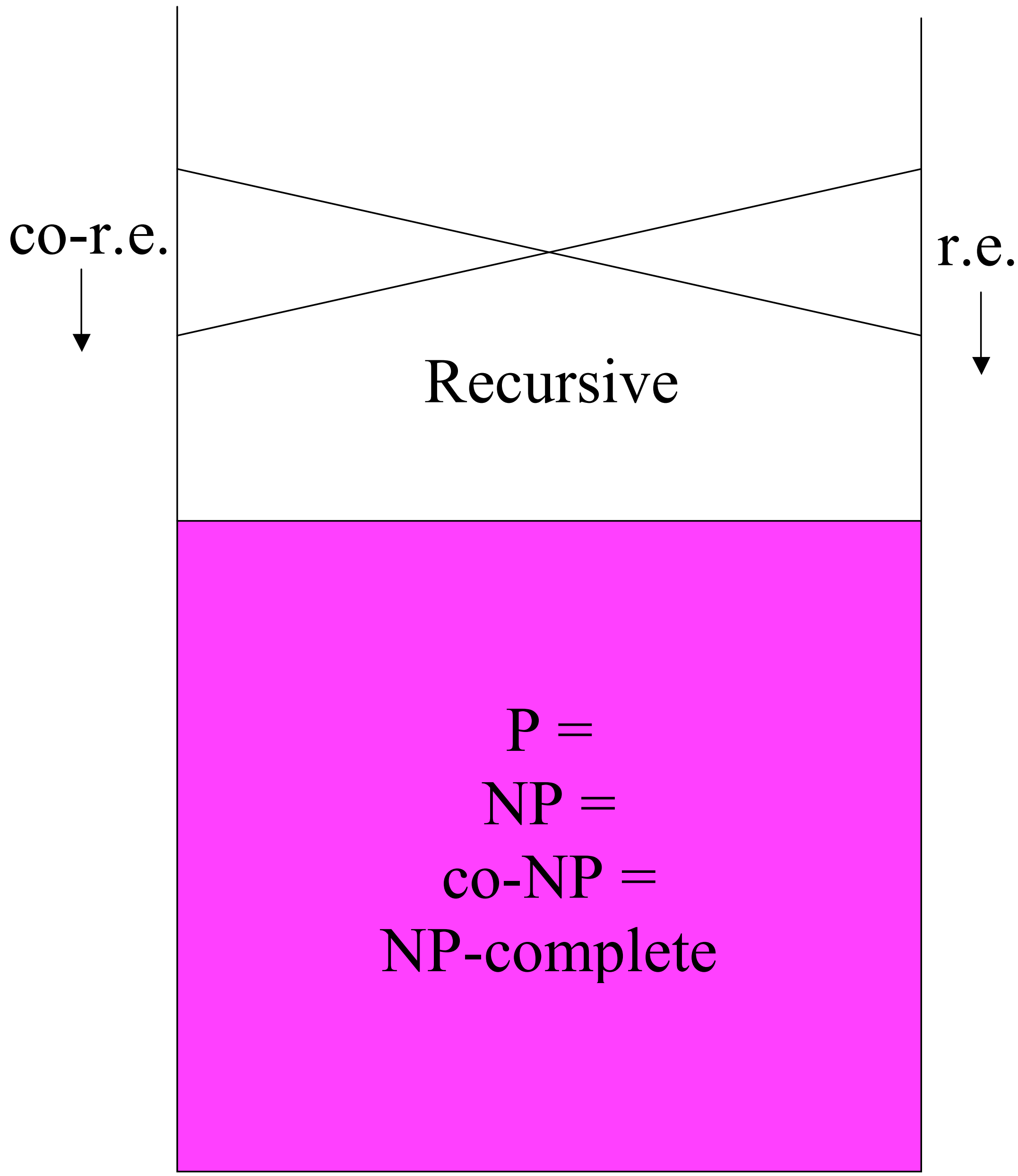
Some natural candidates:

- FACTORING (when described as a language)
- NASH EQUILIBRIUM
- GRAPH ISOMORPHISM
- Any problem in NP $\cap$ co-NP for which we don't know a poly-time algorithm.

The World If P $\neq$ NP



The World If P $=$ NP



Beyond NP

- "Space" as a resource $=$ number of TM squares used in a computation
- A reasonable proxy for memory on other computational models
- PSPACE $=$ languages decidable on TMs using polynomial space
- P $\subseteq$ PSPACE (why?)
- NP $\subseteq$ PSPACE (why?)
- PSPACE $\subseteq$ NPSPACE (why?)

Examples of problems in PSPACE or NPSPACE but probably not in NP

- Determining whether $w \in L(G)$, where G is a context-sensitive grammar, is in NPSPACE
  - Recall that G is a CSG if the right-hand side of every rule is at least as long as its left-had side
  - I.e. if $u \to v$ is a rule then $|u| \leq |v|$
- Determining whether a quantified boolean expression is true is in PSPACE
  - E.g. $\forall x \exists y \forall z [(x \wedge y) \vee (\neg x \wedge \neg z)]$

Savitch's Theorem

NPSPACE $=$ PSPACE

- How much time can a computation take if it uses $O(n^k)$ space and does not loop? $O(2^{n^k})$
- To check deterministically if there exists a computation from TM configuration C_1 to TM configuration C_2 in T steps,
  - for all configurations C, check if
    - there is a computation from C_1 to C of $T/2$ steps and
    - there is a computation from C to C_2 in $T/2$ steps
- To check whether initial configuration yields final configuration takes $O(\log(2^{n^k})) = O(n^k)$ recursion levels and $O(n^k)$ space at each level $\Rightarrow O(n^{2k})$ space

LOGSPACE

A problem is in LOGSPACE if it is decided by a TM that uses only logarithmic space on a work tape and doesn't write on the input tape.

$$\text{LOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE}$$

LOGSPACE $\subsetneq$ PSPACE by a diagonalization argument

So at least one of the three $\subseteq$s must be a $\subsetneq$ but we don't know which!