# More Completeness, coNP, FNP,etc.

CS254

Chris Pollett

Oct 30, 2006.

# Outline

- A last complete problem for NP
- coNP, coNP∩NP
- Function problems

# MAX-CUT

- A **cut** in a graph G=(V,E) is a set of nodes S used to partition G. The **size** of the cut is the number of edges between S and V-S.
- Finding the smallest cut in a graph, MIN-CUT, is in P.
- This is because finding the smallest cut that separates two nodes s and t can be found from our MAX-FLOW algorithm.
- MAX-CUT is the problem given a graph G and a integer k: Is there a cut in G of size k or more?

**Thm.** MAX-CUT is NP-complete.

# Proof

MAX-CUT is in NP because you can always guess a cut and verify it has the desired properties.

To see it is NP-complete we reduce NAE-SAT to it. Given an instance of NAE-SAT, F, we create a graph with 2n nodes corresponding to the variable x1,..xn and their negations. For each clause (a V b V c), we add to E the three edges of the triangle of these literals. If two literals are the same then we omit the edge. Finally, if $x_i$ or its negations occurs n times in F then we add n edges between it and its negation.

(We generalize the notion of graph to allow this.)

Suppose there is a cut of of size 5m where m is the number of clauses. WLOG, we can assume that both a variable and its negation are on the opposite sides of a cut. We can think of the literals in the cut as true and those on the other side as false. The total number of edges in the cut from variables and their negations is thus 3m. The remaining 2m edges must come from cutting triangles. Each triangle must contribute at most two to the cut, so all m triangles must be split. The splitting of a triangle corresponds to two literals in it getting different values.

# NP and coNP

- NP consists of those languages with polynomial length, poly time proofs of membership.
- coNP consists of those languages whose complements are in NP. So languages in it have poly-size, poly-time proofs of non-membership.
- For example, consider the problem VALIDITY. This is the language of encodings of valid propositional formulas.
- Given a formula F, a proof of nonmembership in this language is a satisfying assignment for ¬F.

# coNP-complete

**Prop.** If a language L is NP-complete then its complement is coNP-complete.

**Thm.** VALIDITY is co-NP complete.

**Proof.** It is in coNP, because its complement $\overline{\text{VALIDITY}}$ is in NP (one can guess an assignment and check if it falsifies the formula). $\overline{\text{SAT}}$ is coNP-complete by the previous proposition. A formula F is in this language iff ¬F is valid so $\overline{\text{SAT}}$ reduces to VALIDITY.

**Prop.** If a coNP-complete language is in NP then NP=coNP.

Note: This does not imply P=NP. Call a language L in NP∩coNP if it is in NP and in coNP. Unlike the situation with R= r.e. ∩co-r.e., it is unknown if P is equal to NP∩coNP.

# Function Problems

- So far we have looked at languages.
- Usually we are interested in computing functions. i.e., given a formula we want to find a satisfying assignment. Call this FSAT.
- Notice if we can solve SAT we can solve an FSAT problem. To do this, we can ask for a formula F if it is satisfiable? Then if it is satisfiable we can set a variable to true and ask if the the resulting formula is satisifiable. Repeating, this polynomially many times we can find a satisfying assignment.
- Given a language *L* in NP, we know we can find a polynomially balanced relation $R_L(x,y)$ such that x is in L iff there exists a polynomial length string such that $R_L(x,y)$.
- Let F*L* be the function problem given x find a string y such that $R_L(x,y)$ holds if it exists and return "no" otherwise.
- FNP is the class of all such function problems. FP is the subset of FNP for languages in P. An example, of an FP function is FHornSat.
- We can define complete for these function classes in the same way as we did for language classes.

**Prop.** FP =FNP iff P=NP.

# Total Functions

- We call a problem in FNP **total** if for every string x there is at least one y such that R(x,y).
- We denote by TFNP the function problems in FNP which are total.

Ex: Given an integer N, find its prime factorization $N=p_1^{i\_1}\ldots p_k^{i\_k}$. Since primes is in P, this problem is in TFNP.

Ex: One can show if a cubic graph has a Hamiltonian cycle, that it in fact has a second cycle. Thus, the problem ANOTHER HAMILTONIAN CYCLE on a cubic graph is NP-complete.