Complexity of Counting

Note Title 5/12/2013

So far we have focused on languages (subset of $\{0,1\}^*$).
What about computing functions?

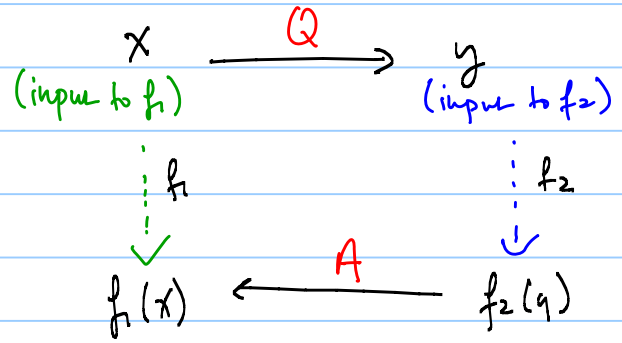
Important type of function: Counting problems.
Given 3-CNF ϕ, how many satisfying assignments does it have?

#P Class of fcns describable as:
input x $\quad f(x) = |\{y \mid (x,y) \in R\}| \quad R \in P$

Compare to NP which just asks: $f(x) > 0$?

example: #SAT: given 3-CNF ϕ how many satisfying assignments?
given (G,k) how many cliques of at least size k?

To talk about reductions for functions, we need to efficiently computable fcns: $f_1 \propto f_2$ if $\exists\, Q, A$



A function f is #P-complete if $f \in$ #P and every $f' \in$ #P $f' \propto f$.

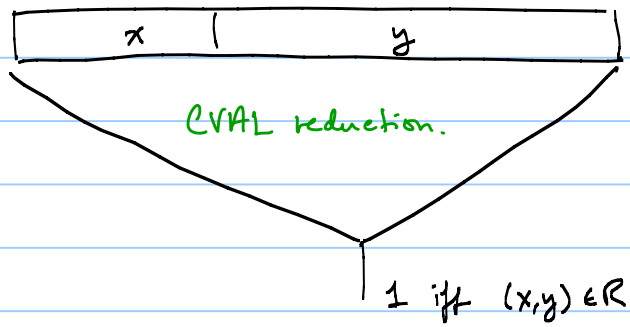
A parsimonious reduction is one in which A is the identity.
The # of witnesses is preserved
* many standard NP-completeness reductions are parsimonious
* if #SAT is #P-complete, we get lots of #P-complete problems.

Theorem: #SAT is #P-complete.

#SAT $\in$ #P $(\phi, A) \in R \iff A$ satisfies ϕ.

Take any $f \in$ #P defined by $R \in P$.

$f(x) = |\{y \mid (x,y) \in R\}|$



In converting circuit to a 3-CNF formula ϕ, we introduce auxiliary variables z.

$$\exists z \; \phi(x,y,z) \iff C(x,y) = 1.$$

The important observation here is that for a fixed y, z_y is unique. $\phi_x(y,z)$ satisfied iff $z_y = z$. $\leftarrow (x,y) \in R$.

So if we hardwire x

$$\#y \text{ s.t. } C(x,y) = 1 = \#(y,z) \text{ s.t. } \phi(x,y,z) = 1. \;//$$

$//$

$|\{y \mid (x,y) \in R\}|$

How to compare #P to other classes?

Use $P^{\#P}$ in order to make it a class of decision problems.

$NP, co\text{-}NP \subseteq P^{\#P}$ ← Can use #P oracle to tell if $|\{y \mid (x,y) \in R\}| = 0$ or ≥ 1

#SAT $\in$ PSPACE so $P^{\#P} \subseteq$ PSPACE.

TODA's Theorem: $PH \subseteq P^{\#P}$ (won't prove here)

He actually proved $PH \subseteq P^{PP}$ → accept if 1/2 of the paths are accepting. (i.e. 1st bit of count = 1).

Is #P hard because it entails finding NP witnesses?
Or is counting difficult by itself?

Definition: $G = (V, U, E)$ $|U| = |V|$ $E \subseteq U \times V$
bipartite graph.

A perfect matching in G is a subset $M \subseteq E$
edges in M touch every node
no two edges in M share an endpoint.

MATCH $\in P$ (given bipartite G, does it contain a perfect matching?)

#MATCH: given bipartite G, how many distinct perfect matchings does it have?

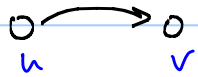
Theorem (Valiant) #MATCH is #P-complete.

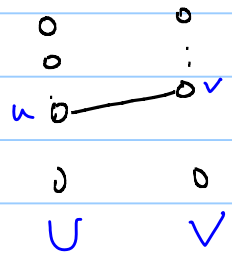
First: #Cycle cover $\propto$ #MATCH parsimonious reduction.

Cycle cover: Input: directed (non-bipartite) graph G.
Is there a set of cycles that visits each vertex exactly once?

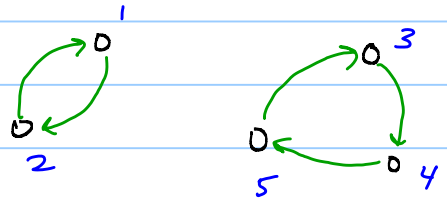
Directed graph G $\longrightarrow$ bipartite G'.

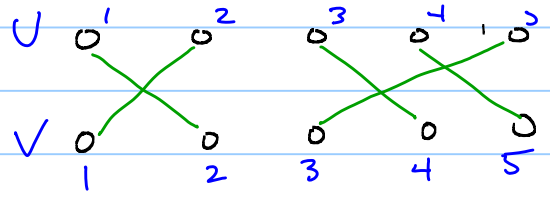
directed edge (u,v) bipartite edge $u \in U$ $v \in V$.





1-1 correspondence between cycle covers & perfect matchings.





Given an $n \times n$ matrix:

$$\text{determinant}(M) = \sum_{\pi} (-1)^{\#\text{swaps}} \sigma(\pi) \prod_{i=1}^{n} A_{i,\pi(i)}$$ $\longrightarrow$ poly-time computable.

$$\text{permanent}(M) = \sum_{\pi} \prod_{i=1}^{n} A_{i,\pi(i)}$$

$$M = \begin{array}{c} \\ U \end{array} \begin{array}{c} \begin{array}{c} V \\ \end{array} \\ \left[\begin{array}{c} 1 \\ 0/1 \end{array} \right. \end{array} \quad \text{Is there an edge } (u,v) \text{ in bipartite } G. \quad = \quad \left[\text{adjacency matrix rep of directed graph } G. \right]$$

perm (M) = # matchings in bipartite graph = # cycle covers in directed G.

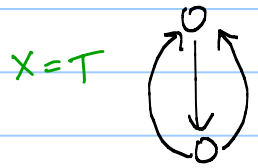
Cycle covers in directed graphs and perfect matchings in bipartite graphs define a permutation over the nodes.

Computing the permanent is the fact that #MATCH is #P-complete means that #P-compl - even on 0/1 graphs.

#SAT ∝ # Cycle Cover.

Note this won't be a parsimonious reduction or else P=NP!

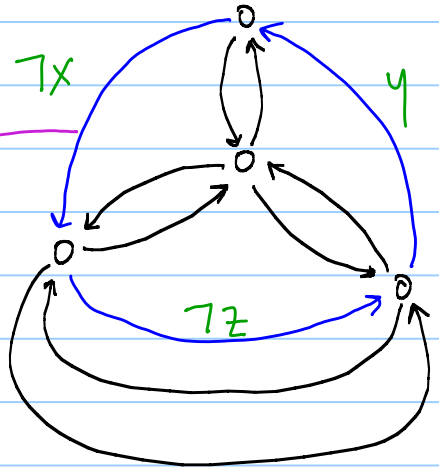
Gadget for variables:



need a gadget to enforce that exactly one edge is chosen

Cycle chosen determines truth value of X.

Gadget for clauses:



Each blue edge represents a literal in the clause. The blue edge is in the cycle cover iff the literal is false. (will need to find a way to enforce this).

There is no cycle cover of the clause gadget which includes all three blue edges. (i.e. can't have all literals in the clause be false).

For any proper subset of the blue edges, there is exactly one cycle cover of the clause gadget.

If we can find a way to enforce the X-or condition, then the # Sat assignments = # cycle covers.

Exclusive OR gadget:

Consider a version of cycle cover with weights on the edges.

Want to compute: $\sum_{\text{cycle covers}}$ wt(cycle cover)

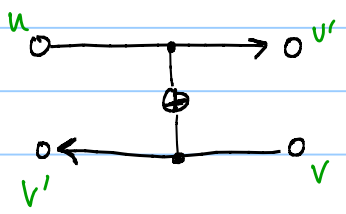
↳ product of edge weights (non-edges implicitly have wt 0).

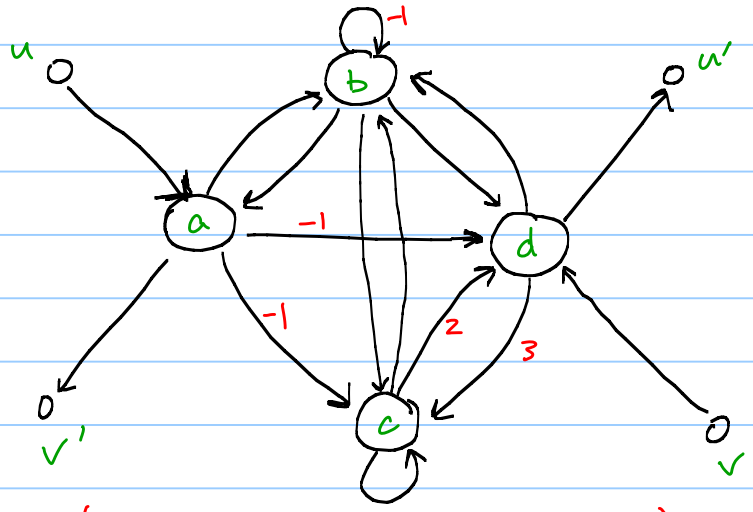
If all edges have wt 1, then this is the same as # Cycle cover. We will show:

#SAT $\propto$ # (Weighted Cycle Cover) $\propto$ # Cycle Cover mod N $\propto$ # Cycle Cover

(easy.)

First show X-or gadget with weights (#SAT $\propto$ #(WT-CYCLE-COVER))





(unlabeled edges have weight = 1)

	a	b	c	d
a	0	1	-1	-1
b	1	-1	1	1
c	0	1	1	2
d	0	1	3	0

a) permanent is 0 (i.e. can't omit both (u,u') (v,v') edges)

b) if rows + cols for a + d are deleted, perm = 0.
Can't go in by u + out by v'.
And in by v and out by u'.

c) if we delete 1st row + 1st col, perm = 0.
(in by v + out by v')

d) if we delete 4th row + col, perm = 0.
(in by v + out by v').

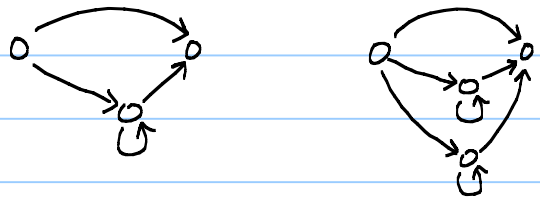
⇒ the only cycle covers that contribute to the permanent are ones that go in by v + out by v' (mult by 4)
OR in by v' + out by v. (mult by 4)

If we divide the graph into two components C + C' and consider only cycles covers w/ cycles within C or within C', then the # cycle covers =
Perm (C) · Perm (C').
if one of these is 0, then these cycle covers don't contribute to overall Perm (G).

⇒ #(wt-cycle cover) = (# Sat assignment) 4^{3m} ← # of x-or gadgets.

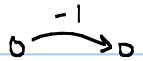
How to simulate the non 0/1 entries?
#(wt-cycle cover) & #(Cycle cover).

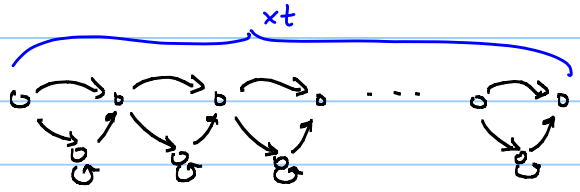
For wt = 2 or 3:



What about -1? pick $N = 2^t + 1$.

edges will have weight $2^t \equiv -1 \bmod N$.

Replace  with:



pick $t = 8m$

$$\sum_{\substack{\text{cycle} \\ \text{covers } C}} \prod_{\substack{\text{edges} \\ \text{in} \\ C}} w(e) = \sum_{C} wt'(C)\,(N-1)^{r_C}$$

$wt'(C)$: wt from hm $N-1$ edges.

r_C = # of $N-1$ edges.

$$\left[\sum_{C} wt'(C)\,(N-1)^{r_C}\right] \bmod N = \sum_{C} wt'(C)\,(-1)^{r_C}$$

as long as $\sum_{C} wt'(C)\,(-1)^{r_C} < N = 2^{8m}$.

$\leq 2^n \cdot 4^{3m} < 2^m \cdot 2^{6m} = 2^{7m}$

2^n: # sat assign