Complexity of Counting

What is the complexity of counting the number of:

1. Satisfying assignments of a given cnf/CNF. (#SAT)
2. Hamiltonian cycle in a graph. (#HamCycle) → undirected graph.
3. Perfect matching in a bipartite graph. (#PerfectMatchg)
4. Cycles in a digraph. (#Cycles)
5. Spanning trees in a connected graph. (#SpTrees)
6. Simple paths between two vertices s & t in a connected graph. (#Paths)

Obs: Prob. 1 & 2 are NP-hard.

Prob 3 & 4 : The corresponding decision problems are in P.

Prob 5 & 6 : " " " " are trivial.

Thm: (by Kirchhoff's thm) Prob 5 is in FP.

Laplacian matrix of a graph: Let G be a connected graph without self loops.

L is a $n \times n$ matrix.

$L_{ij} = d_{ii}$ if $i = j$ (d_{ii} = deg of the i-th vertex)

$= -1$ if there's an edge from i to j

$= 0$ o/w.

$L = D - A$

D → deg matrix

A → adj matrix

Obs: Easy to compute L fr A.

Thm (Kirchhoff 1847):

sp. trees in $G = \det\left(L^{(ii)}\right)$ for any i.

$L^{(ii)}$ = the minor obtained from L by removing the (i,i)-th entry of L.

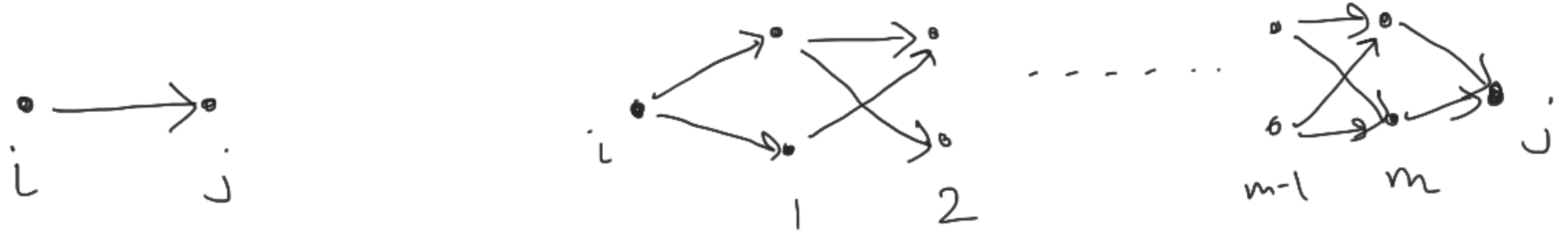
Cor: Prob. $\overline{5}$ in in (functional NC.

Prob 9: #Cycle.

Thm: If #Cycle is in P, then NP = P.

Pf: Let G be a digraph. We'll form a new graph G' from G s.t. if we could count the # cycles in G' efficiently then we can also infer if G has a ham. cycle efficiently.

vertices in G = n



G

G'

Case 1: G has a ham. cycle.

$$\#\text{cycles}(G') \geq 2^{m \cdot n}.$$

Case 2: G has no ham. cycle. $\Rightarrow \#\text{cycles}(G) \leq n^{n-1}$.

$$\#\text{cycles}(G') \leq n^{n-1} \cdot 2^{m(n-1)}.$$

If we choose m in such a way that

$$n^{n-1} \cdot 2^{m(n-1)} < 2^{m \cdot n}$$

$$\Leftrightarrow n^{n-1} < 2^{m} \quad \Rightarrow m > (n-1)\log n.$$

Set $m = n^2$. ∎

Moral: Complexity of counting can be quite high even if the corresponding decision prob is in P.

Class #P :

Defn: We say a function $f: \{0,1\}^* \to \{0,1\}^*$ is in #P if there's a poly-time DTM M & a poly. function $q(\cdot)$ s.t. for every $x \in \{0,1\}^*$,

$$f(x) = \left| \{ y \in \{0,1\}^{q(|x|)} : M(x,y) = 1 \} \right|.$$

Obs: Prob 1-6 are all in #P.

#P-completeness

Q. Is #P = FP?

Defn: A function f is #P-complete if $f \in \#P$ and for every $g \in \#P$, $g \in FP^f$.

for every $x \in \{0,1\}^*$, we can compute $g(x)$ in poly-time using oracle access to f.

f is used as an oracle to poly-tm DTM.

Obs: If a #P-complete language is in FP, then $\#P = FP$.

Thm: Prob1 (#SAT) is #P-complete.

Pf: Let $g \in \#P$. **Goal:** $g \in FP^{\#SAT}$.

There's a poly-time DTM M & a poly-fn $q(\cdot)$ s.t. for every x.

$$g(x) = \left| \{ y \in \{0,1\}^{q(|x|)} : M(x,y) = 1 \} \right|$$

$\updownarrow$ (Cook-Levin)

$\psi_x(y) = 1$

- Give ψ_x as i/p to the oracle #SAT.
- (ψ_x may not be a CNF) ∎

#SAT $\in$ #P :

Note: It is not known if the counting version of every NP-complete prob is #P-complete. [issue: reduction needn't be parsimonious]

Thm: Prob 2 (#HamCycle) is #P-complete.

What abt Prob 3 & 6?

Thm (Valiant 1979): Prob 6 is also #P-complete.

Ref: "The Complexity of Enumeration & Reliability Problems" by Valiant (1979).

- #PATH is #P-complete for both directed & undirected graph.

Thm (Valiant 1979): Prob 3 is also #P-complete.

Pf: We'll see a proof in the next lecture.

Ref: "The complexity of computing Permanent".

Obs1: $\#P \subseteq PSPACE$. (by defn)

- $PH \subseteq PSPACE$.
  - How does #P relate to PH?

Thm: (Toda 1991): $PH \subseteq P^{\#SAT}$.

Pf: We'll prove it later.

▷ #P is "harder" than PH.

Ref: "PP is as hard as the PH" by Toda (1991).

$\sim$ ↓ ?? (probabilistic poly-time)

Obs 2: If #P = FP, then P = NP.

How about the converse?

▷ If P = NP, then is #P = FP? We don't know.

▷ But, we do know that if P = NP then every #P problem has a randomized poly-time approximation algo.

↳ Interesting
- can be derandomized

Defn: A function $f: \{0,1\}^* \to \{0,1\}^*$ has a fully polytime randomized approximation scheme (FPRAS) if for every $\epsilon, \delta > 0$, there's a PTM M s.t. for every $x \in \{0,1\}^*$,

- $(1-\epsilon) f(x) \leq M(x) \leq (1+\epsilon) f(x)$ w.p. $\geq 1-\delta$
- $M(x)$ runs in $\text{poly}\left(|x|, \frac{1}{\epsilon}, \log \frac{1}{\delta}\right)$.

Thm: If $P = NP$ then every #P function has a FPRAS.

Pf: We'll prove it later.

Remark: In fact, in the above thm FPRAS can be replaced by FPTAS (fully poly-time approx. scheme).

Some #P-complete problems do admit FPRAS unconditionally.

e.g. #Perfect Matching.

Thm: (Jerrum, Sinclair, Vigoda 2001)
Permanent of an $n \times n$ matrix with non-negative entries has an FPRAS.

Note: No derandomized algo is known!

Ref:- "A poly-time approx. algo for the perm. of a matrix with non-negative entries"