Computational Complexity Theory

Lecture 23: Complexity of Counting

Department of Computer Science,
Indian Institute of Science

Natural counting problems

- What is the complexity of the following problems?

- #SAT: Count the number of satisfying assignments of a given Boolean circuit/CNF.

- #HAMCYCLE: Count the number of Hamiltonian cycles in an undirected graph.

- Observation. The above problems are NP-hard.

Natural counting problems

- What is the complexity of the following problems?

- #PerfectMatching: Count the number of perfect matchings in a bipartite graph.

- #CYCLE: Count the number of simple cycles in a directed graph.

- Observation. The corresponding decision problems are in P.

Natural counting problems

- What is the complexity of the following problems?

- #PATH: Count the number of simple paths between two vertices in a connected graph.

- #SPANTREE: Count the number of spanning trees in a connected graph.

- Observation. The corresponding decision problems are trivial.

An easy counting problem

- Theorem. *(Kirchhoff 1847)* #SPANTREE is in FP.

An easy counting problem

- Theorem. *(Kirchhoff 1847)* #SPANTREE is in FP.

- Proof sketch. Let G be an n-vertex connected graph without self loops. Label the vertices by $\{1,\ldots, n\}$.

- Definition. The *Laplacian matrix* of G is an $n \times n$ matrix L_G defined as

$$
\begin{aligned}
L_G(i,j) &= \deg(i) && \text{if } i = j, \\
&= -1 && \text{if there's an edge } (i,j) \text{ in } G, \\
&= 0 && \text{otherwise.}
\end{aligned}
$$

An easy counting problem

- Theorem. *(Kirchhoff 1847)* #SPANTREE is in FP.

- Proof sketch. Let G be an n-vertex connected graph without self loops. Label the vertices by $\{1,\ldots, n\}$.

- Definition. The *Laplacian matrix* of G is an $n \times n$ matrix L_G defined as $L_G = D_G - A_G$, where D_G is the degree matrix and A_G the adjacency matrix of G.

- Observation. It is easy to compute L_G from A_G.

An easy counting problem

- Theorem. *(Kirchhoff 1847)* #SPANTREE is in FP.

- Proof sketch. Let G be an n-vertex connected graph without self loops. Label the vertices by $\{1,\ldots, n\}$.

- <u>*Kirchhoff's matrix-tree theorem*</u> states that

  no. of spanning trees of G = any cofactor of L_G.

- (i,j) *cofactor* of $L = (-1)^{i+j} \cdot \det($submatrix of L obtained by deleting the i-th row and the j-th column from $L)$.

An easy counting problem

- Theorem. *(Kirchhoff 1847)* #SPANTREE is in FP.

- Proof sketch. Let G be an n-vertex connected graph without self loops. Label the vertices by $\{1, \ldots, n\}$.

- *Kirchhoff's matrix-tree theorem* states that

  no. of spanning trees of G = any cofactor of L_G. ∎

- Corollary. As determinant computation is in (functional) NC, #SPANTREES is in (functional) NC.

A hard counting problem

- Theorem. #CYCLE is in NP-hard.
- Lesson. A counting problem can be hard even if the corresponding decision problem is in P.

A hard counting problem

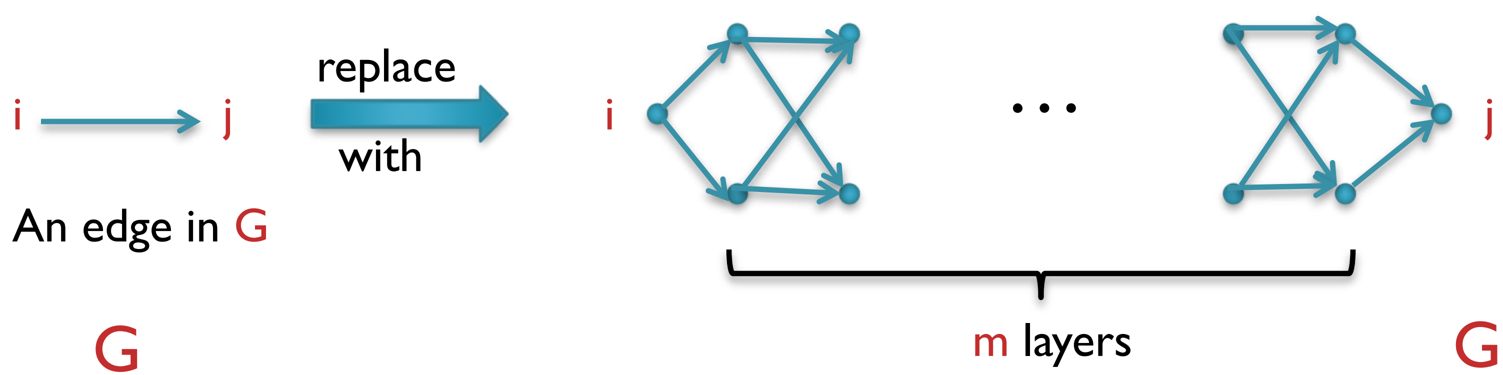
- Theorem. #CYCLE is in NP-hard.

- Proof. We will give a poly-time reduction from the Hamiltonian cycle problem to the #CYCLE problem.

A hard counting problem

- Theorem. #CYCLE is in NP-hard.

- Proof. Let G be an n-vertex digraph. We'll efficiently construct a new graph G' from G s.t. the presence of a Hamiltonian cycle in G can be readily derived from the number of cycles in G'. Construction of G' :

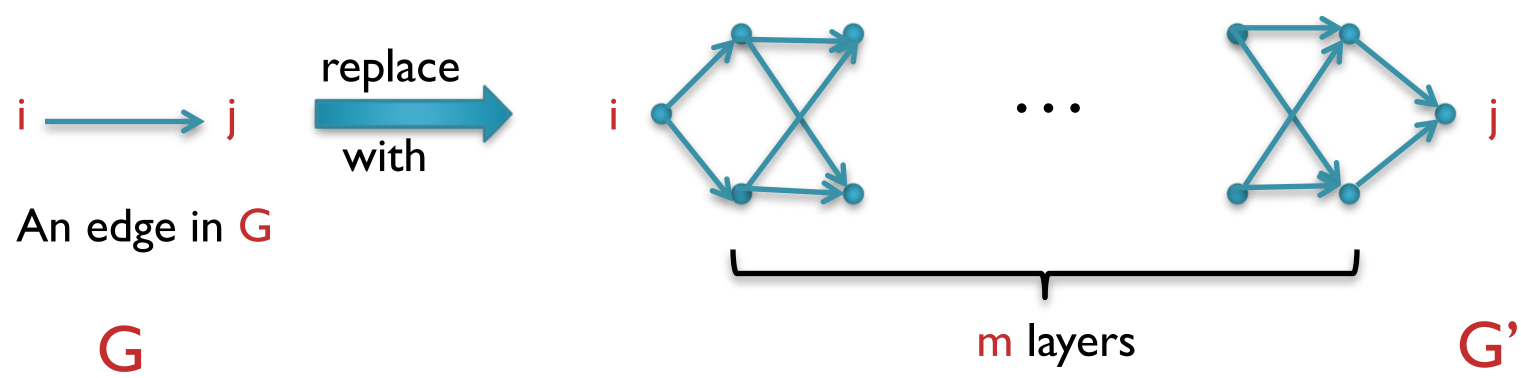


A hard counting problem

- Theorem. #CYCLE is in NP-hard.

- Proof. Case1: If G has a HC, then #cycle(G') $\geq 2^{mn}$.

#cycle → no. of cycles

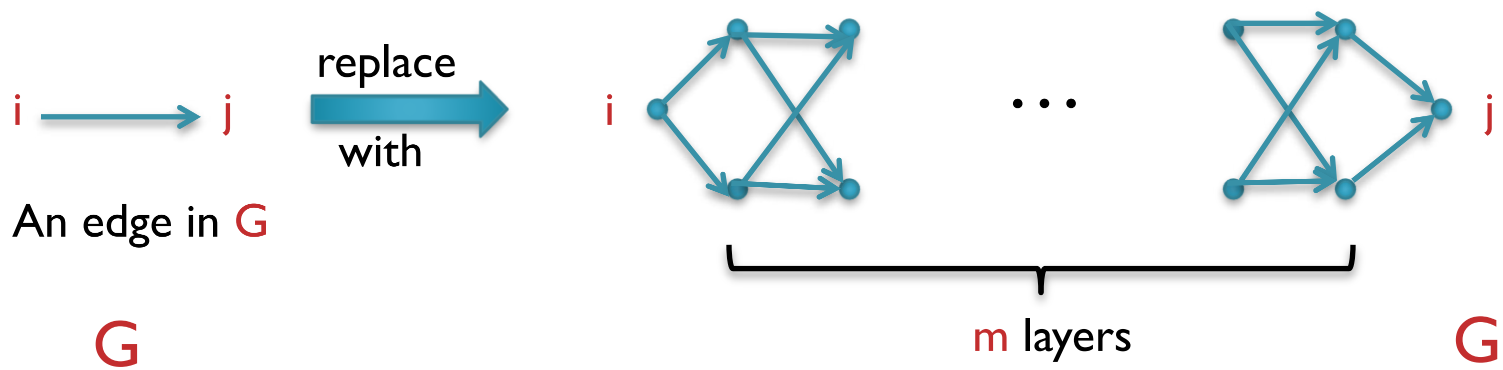


A hard counting problem

- Theorem. #CYCLE is in NP-hard.

- Proof. Case1: If G has a HC, then #cycle(G') ≥ 2^{mn}.
- Case2: If G has no HC, then #cycle(G) ≤ n^{n-1}

  #cycle(G') ≤ $n^{n-1}.2^{m(n-1)}$.



A hard counting problem

- Theorem. #CYCLE is in NP-hard.

- Proof. Case1: If G has a HC, then $\#cycle(G') \geq 2^{mn}$.
- Case2: If G has no HC, then $\#cycle(G) \leq n^{n-1}$

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

- If we choose m such that $n^{n-1}.2^{m(n-1)} < 2^{mn}$, then we can find out if G has a HC from $\#cycle(G')$.
- Set $m = n^2$.

Class #P

- Definition. We say a function $f: \{0,1\}^* \rightarrow \mathbb{N}$ is in #P if there's a poly-time TM M and a polynomial function $p: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

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

Class #P

- Definition. We say a function $f: \{0,1\}^* \to \mathbb{N}$ is in #P if there's a poly-time TM M and a polynomial function $p: \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

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

- Observation. Problems #SAT, #HAMCYCLE, #PerfectMatching, #CYCLE, #PATH and #SPANTREE are in #P.
- In fact, with every language in NP we can associate a counting problem that is in #P.

#P-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is #P = FP ?

#P-completeness

- Definition. A function $f: \{0,1\}^* \rightarrow N$ is in #P-complete if f is in #P and for every $g \in$ #P, we have $g \in FP^f$ i.e., g is poly-time Cook/Turing reducible to f.

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

#P-completeness

- Definition. A function $f: \{0,1\}^* \rightarrow N$ is in #P-complete if f is in #P and for every $g \in$ #P, we have $g \in FP^f$ i.e., g is poly-time Cook/Turing reducible to f.

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

- Observation. If a #P-complete language is in FP then #P = FP.

Natural #P-complete problems

- Theorem. #SAT is #P-complete.

- Proof. #SAT is in #P. Let $g \in$ #P. We intend to show that $g \in \mathrm{FP}^{\#\mathrm{SAT}}$.

Natural #P-complete problems

- Theorem. #SAT is #P-complete.

- Proof. #SAT is in #P. Let $g \in$ #P. We intend to show that $g \in \mathrm{FP}^{\#\mathrm{SAT}}$. There's a poly-time TM M and a poly. function $p: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

$$g(x) = \left|\{u \in \{0,1\}^{p(|x|)} : M(x, u) = 1\}\right| .$$

- Algorithm: On input x, convert $M(x, ..)$ to a 3CNF ϕ_x using Cook-Levin theorem. Give ϕ_x as input to the #SAT oracle. Output whatever the oracle outputs.

Natural #P-complete problems

- Theorem. #SAT is #P-complete.

- Proof. #SAT is in #P. Let $g \in \#P$. We intend to show that $g \in FP^{\#SAT}$. There's a poly-time TM M and a poly. function $p: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

$$g(x) = \left|\{u \in \{0,1\}^{p(|x|)} : M(x, u) = 1\}\right| .$$

- Algorithm: On input x, convert $M(x, ..)$ to a 3CNF ϕ_x using Cook-Levin theorem. Give ϕ_x as input to the #SAT oracle. Output whatever the oracle outputs.

Note: Only <u>one query</u> to the oracle. Resembles a poly-time Karp reduction.

Natural #P-complete problems

- Theorem. #SAT is #P-complete.

- Proof. #SAT is in #P. Let $g \in$ #P. We intend to show that $g \in \mathrm{FP}^{\#\mathrm{SAT}}$. There's a poly-time TM M and a poly. function $p: \mathbb{N} \rightarrow \mathbb{N}$ such that for every $x \in \{0,1\}^*$,

$$g(x) = \left|\{u \in \{0,1\}^{p(|x|)} : M(x,u) = 1\}\right| .$$

- Correctness: Follows from the fact that the Cook-Levin reduction is <u>*parsimonious*</u>, i.e.,

$$\left|\{u \in \{0,1\}^{p(|x|)} : M(x,u) = 1\}\right| = \#\phi_x .$$

The no. of satisfying assignments of ϕ_x.

Natural #P-complete problems

- Theorem. #HAMCYCLE is #P-complete.

- Most (all?) NP-complete problems known till date have defining verifiers such that the corresponding counting problems are #P-complete.

- Open. Does every NP-complete problem have a defining verifier such that the corresponding counting problem is #P-complete ?

Issue: The reduction that shows NP-completeness of a problem needn't have to be parsimonious.

Natural #P-complete problems

- Theorem. *(Valiant 1979)* #PATH is #P-complete.

- In fact, #PATH is #P-complete for both directed and undirected graphs.

Natural #P-complete problems

- Theorem. *(Valiant 1979)* #PATH is #P-complete.
- In fact, #PATH is #P-complete for both directed and undirected graphs.
- Theorem. *(Valiant 1979)* #PerfectMatching is #P-complete.
- Proof. We'll see a proof later.

Relation between #P and other classes

- Observation. $\#P \subseteq PSPACE$.

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

Relation between #P and other classes

- Observation. #P $\subseteq$ PSPACE.

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

- Theorem. *(Toda 1991)* PH $\subseteq$ $P^{\#SAT}$.
- Proof. We'll see a proof later.

Relation between #P and other classes

- Observation. $\#P \subseteq PSPACE$.
- Also, $PH \subseteq PSPACE$. How does #P relate to PH ?
- Theorem. *(Toda 1991)* $PH \subseteq P^{\#SAT}$.
- Hence, #P is *harder* than PH.

Approximations of #P functions

- Observation. If #P = FP, then P = NP.
- Open. Does P = NP imply #P = FP ?

- But, we do know that P = NP implies every #P problem has a randomized polynomial-time *approximation* algorithm.

Approximations of #P functions

- Observation. If #P = FP, then P = NP.
- Open. Does P = NP imply #P = FP ?

- But, we do know that P = NP implies every #P problem has a <u>randomized</u> polynomial-time approximation algorithm.

Can be derandomized!

Approximations of #P functions

- Definition. A function $f: \{0,1\}^* \longrightarrow \mathbb{N}$ has a *Fully Polynomial-time Randomized Approximation Scheme* (FPRAS) if for every $\varepsilon, \delta > 0$, there's a PTM M such that for every $x \in \{0,1\}^*$,
  - $(1-\varepsilon).f(x) \leq M(x) \leq (1+\varepsilon).f(x)$ with prob. $\geq 1 - \delta$,
  - M runs in $\mathrm{poly}(|x|, \varepsilon^{-1}, \log \delta^{-1})$ time.

Approximations of #P functions

- Definition. A function $f: \{0,1\}^* \rightarrow \mathbb{N}$ has a *Fully Polynomial-time Randomized Approximation Scheme* (FPRAS) if for every $\varepsilon, \delta > 0$, there's a PTM M such that for every $x \in \{0,1\}^*$,
  - $(1-\varepsilon).f(x) \leq M(x) \leq (1+\varepsilon).f(x)$ with prob. $\geq 1-\delta$,
  - M runs in $\mathrm{poly}(|x|, \varepsilon^{-1}, \log \delta^{-1})$ time.

- Theorem. If $P = NP$ then every #P function has a FPRAS.
- Proof. We'll see a proof later.

Approximations of #P functions

- Definition. A function $f: \{0,1\}^* \rightarrow \mathbb{N}$ has a *Fully Polynomial-time Randomized Approximation Scheme* (FPRAS) if for every $\varepsilon, \delta > 0$, there's a PTM M such that for every $x \in \{0,1\}^*$,
  - $(1-\varepsilon).f(x) \leq M(x) \leq (1+\varepsilon).f(x)$ with prob. $\geq 1-\delta$,
  - M runs in $poly(|x|, \varepsilon^{-1}, \log \delta^{-1})$ time.

- Theorem. If P = NP then every #P function has a FPRAS.
- Remark. In fact the above FPRAS can be replaced by a FPTAS (*Fully Poly-Time Approximation Scheme*).

Approximations of #P functions

- Some #P-complete problems do admit FPRAS <u>unconditionally</u>!

- Theorem. *(Jerrum, Sinclair, Vigoda 2001)* #PerfectMatching has a FPRAS.

- Remark. No derandomization of this algorithm is known!

Approximations of #P functions

- Some #P-complete problems do admit FPRAS <u>unconditionally</u>!

- Theorem. *(Jerrum, Sinclair, Vigoda 2001)* Permanent of a square matrix with non-negative entries has a FPRAS.

- If $X = (x_{ij})_{i,j \in n}$ then $\mathrm{Perm}(X) = \sum_{\sigma \in S_n} \prod_{i \in [n]} x_{i\,\sigma(i)}$.

Approximations of #P functions

- Some #P-complete problems do admit FPRAS <u>unconditionally</u>!

- Theorem. *(Jerrum, Sinclair, Vigoda 2001)* Permanent of a square matrix with non-negative entries has a FPRAS.

- If $X = (x_{ij})_{i,j \in n}$ then $\mathrm{Perm}(X) = \sum_{\sigma \in S_n} \prod_{i \in [n]} x_{i\,\sigma(i)}$.

- Note. If B_G is the biadjacency matrix of a bipartite graph G, then $\mathrm{Perm}(B_G) = \#\mathrm{PerfectMatchings}(G)$.

0/1 matrix

0/1-Permanent is #P-complete

- Theorem. *(Valiant 1979)* 0/1-Perm is #P-complete.

- It implies that #PerfectMatchings is #P-complete.

0/1-Permanent is #P-complete

- Theorem. *(Valiant 1979)* 0/1-Perm is #P-complete.

- Proof. 0/1-Perm is in #P. (Why?)

0/1-Permanent is #P-complete

- Theorem. *(Valiant 1979)* 0/1-Perm is #P-complete.
- Proof. We'll show that $\#3SAT \in FP^{0/1\text{-Perm}}$.
- In fact, we'll give a poly-time "Karp-like" reduction from #3SAT to 0/1-Perm, i.e., we'll give a poly-time computable function that maps a 3CNF ϕ to a 0/1-matrix A_ϕ s.t. $\#\phi$ is efficiently computable from $A\phi$.
- This means only <u>one query</u> to the 0/1-Perm oracle is required.

…the proof will be given in the next lecture

Graph theoretic interpretation of Perm

- Let $A = (a_{ij})_{i,j \in r}$, where $a_{ij} \in R$.
- Then, $\mathrm{Perm}(A) = \sum_{\sigma \in S_r} \prod_{i \in [r]} a_{i\,\sigma(i)}$.

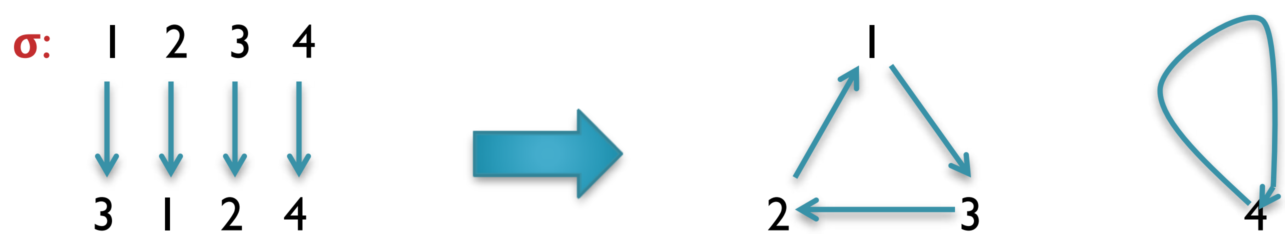
- Let G be the weighted digraph on r vertices with adjacency matrix A, i.e., the edge (i, j) in G has weight a_{ij}.

Graph theoretic interpretation of Perm

- Let $A = (a_{ij})_{i,j \in r}$, where $a_{ij} \in \mathbb{R}$.
- Then, $\mathrm{Perm}(A) = \sum_{\sigma \in S_r} \prod_{i \in [r]} a_{i\,\sigma(i)}$.

- Let G be the weighted digraph on r vertices with adjacency matrix A, i.e., the edge (i, j) in G has weight a_{ij}.

- Every permutation σ: $[r] \longrightarrow [r]$ can be expressed (uniquely) as a product of disjoint cycles.



Graph theoretic interpretation of Perm

- Definition. A *cycle cover* of a digraph G is a subgraph of G having in-degree and out-degree of every vertex exactly 1, i.e., the subgraph is a disjoint union of cycles covering all the vertices of G.

- *Weight* of a cycle cover C, denoted wt(C), is defined as the product of the weights of the edges in C.

Graph theoretic interpretation of Perm

- Definition. A <u>*cycle cover*</u> of a digraph G is a subgraph of G having in-degree and out-degree of every vertex exactly 1, i.e., the subgraph is a disjoint union of cycles covering all the vertices of G.

- <u>*Weight*</u> of a cycle cover C, denoted wt(C), is defined as the <u>product</u> of the weights of the edges in C.

- Observation. $\mathrm{Perm}(A) = \sum_{\substack{C:\ C \text{ is Cycle} \\ \text{cover of } G}} \mathrm{wt}(C)$.