Complexity of Counting

Antonis Antonopoulos

CoReLab
NTUA

March 2014

Why counting?

- So far, we have seen two versions of problems:
  - Decision Problems (if a solution *exists*)
  - Function Problems (if a solution can be *produced*)
- A very important type of problems in Complexity Theory is also:
  - Counting Problems (*how many* solution exist)

Example (#SAT)

Given a Boolean Expression, compute the number of different truth assignments that satisfy it.

- Note that if we can solve #SAT in polynomial time, we can solve SAT also.
- Similarly, we can define #HAMILTON PATH, #CLIQUE, etc.

Basic Definitions

Definition (#P)

A function $f : \{0,1\}^* \to \mathbb{N}$ is in $\#\mathbf{P}$ if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing Machine M such that for every $x \in \{0,1\}^*$:

$$f(x) = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$$

- The definition implies that $f(x)$ can be expressed in $poly(|x|)$ bits.
- Each function f in $\#\mathbf{P}$ is equal to the number of paths from an initial configuration to an accepting configuration, or **accepting paths** in the configuration graph of a poly-time NDTM.
- $\mathbf{FP} \subseteq \#\mathbf{P} \subseteq \mathbf{PSPACE}$
- If $\#\mathbf{P} = \mathbf{FP}$, then $\mathbf{P} = \mathbf{NP}$.
- If $\mathbf{P} = \mathbf{PSPACE}$, then $\#\mathbf{P} = \mathbf{FP}$.

- In order to formalize a notion of completeness for $\#\mathbf{P}$, we must define proper reductions:

Definition (Cook Reduction)

A function f is $\#\mathbf{P}$-complete if it is in $\#\mathbf{P}$ and every $g \in \#\mathbf{P}$ is in $\mathbf{FP}^g$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in \mathbf{NP}$, then $\#A(x) = |\{y \in \{0,1\}^{p(|x|)} : \ R_A(x,y) = 1\}| \in \#\mathbf{P}$

- In order to formalize a notion of completeness for #**P**, we must define proper reductions:

Definition (Cook Reduction)

A function f is #**P**-complete if it is in #**P** and every $g \in \#\mathbf{P}$ is in $\mathbf{FP}^g$.

- As we saw, for each problem in **NP** we can define the associated counting problem: If $A \in \mathbf{NP}$, then $\#A(x) = |\{y \in \{0,1\}^{p(|x|)} : \ R_A(x,y) = 1\}| \in \#\mathbf{P}$
- We now define a more strict form of reduction:

Definition (Parsimonious Reduction)

We say that there is a parsimonious reduction from $\#A$ to $\#B$ if there is a polynomial time transformation f such that for all x:

$$|\{y : \ R_A(x,y) = 1\}| = |\{z : \ R_B(f(x),z) = 1\}|$$

Completeness Results

Theorem

$\#\texttt{CIRCUIT SAT}$ *is* $\#\mathbf{P}$*-complete.*

Proof:

- Let $f \in \#\mathbf{P}$. Then, $\exists M, p$:
  $f = |\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}|$.
- Given x, we want to construct a circuit C such that:

$$|\{z : \ C(z)\}| = |\{y : \ y \in \{0,1\}^{p(|x|}, M(x,y) = 1\}|$$

- We can construct a circuit $\hat{C}$ such that on input x, y simulates $M(x,y)$.
- We know that this can be done with a circuit with size about the square of M's running time.
- Let $C(y) = \hat{C}(x,y)$. □

Completeness Results

Theorem

$\#\texttt{SAT}$ *is* $\#\mathbf{P}$*-complete.*

Proof:

- We reduce $\#\texttt{CIRCUIT SAT}$ to $\#\texttt{SAT}$:
- Let a circuit C, with $x_1, \ldots, x_n$ input gates and $1, \ldots, m$ gates.
- We construct a Boolean formula ϕ with variables $x_1, \ldots, x_n, g_1, \ldots, g_m$, where g_i represents the output of gate i.
- A gate can be complete described by simulating the output for each of the 4 possible inputs.
- In this way, we have reduced C to a formula ϕ with $n + m$ variables and $4m$ clauses. □

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_i\} \in E$ iff $A_{i,j} = 1$.

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0,1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_i\} \in E$ iff $A_{i,j} = 1$.
- The term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ has a perfect matching.

The Permanent

Definition (PERMANENT)

For a $n \times n$ matrix A, the permanent of A is:

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- Permanent is similar to the determinant, but it seems more difficult to compute.
- Combinatorial interpretation: If A has entries $\in \{0, 1\}$, it can be viewed as the adjacency matrix of a bipartite graph $G(X, Y, E)$ with $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and $\{x_i, y_i\} \in E$ iff $A_{i,j} = 1$.
- The term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ is 1 iff σ has a perfect matching.
- So, in this case $perm(A)$ is the number of perfect matchings in the corresponding graph!

Valiant's Theorem

Theorem (Valiant's Theorem)

*PERMANENT is #**P**-complete.*

- Notice that the decision version of `PERMANENT` is in **P** ! !

Quantifiers vs Counting

- An imporant open question in the 80s concerned the relative power of Polynomial Hierarchy and $\#\mathbf{P}$.
- Both are natural generalizations of **NP**, but it seemed that their features were not directly comparable to each other.
- But, in 1989, S. Toda showed the following theorem:

Quantifiers vs Counting

- An imporant open question in the 80s concerned the relative power of Polynomial Hierarchy and $\#\mathbf{P}$.
- Both are natural generalizations of **NP**, but it seemed that their features were not directly comparable to each other.
- But, in 1989, S. Toda showed the following theorem:

Theorem (Toda's Theorem)

$$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}[1]}$$

The Class $\oplus\mathbf{P}$

Definition

A language L is in the class $\oplus\mathbf{P}$ if there is a NDTM M such that for all strings x, $x \in L$ iff the *number of accepting paths* on input x is odd.

- The problems $\oplus$SAT and $\oplus$HAMILTON PATH are $\oplus\mathbf{P}$-complete.
- $\oplus\mathbf{P}$ is closed under complement.

Theorem

$$\mathbf{NP} \subseteq \mathbf{RP}^{\oplus\mathbf{P}}$$

The Class GapP

- For a TM M, we define:

$$\Delta M(x) = acc(x) - rej(x) = \#M(x) - \#\overline{M}(x)$$

Definition

A function $f : \{0,1\}^* \to \mathbb{N}$ is in **GapP** if there exists a poly-time NDTM M such that for all inputs x:

$$f(x) = \Delta M(x)$$

- **GapP** functions are **closed under negation**: $f \in$ **GapP** $\Rightarrow -f \in$ **GapP**.
- **GapP**, unlike $\#$**P**, encompasses <u>all</u> **FP** functions.

The Class GapP

Theorem

For all functions f, the following are equivalent:

1. $f \in \mathbf{GapP}$.
2. f *is the difference of two* $\#\mathbf{P}$ *functions.*
3. f *is the difference of a* $\#\mathbf{P}$ *and an* $\mathbf{FP}$ *function.*
4. f *is the difference of a* $\mathbf{FP}$ *and an* $\#\mathbf{P}$ *function.*

In other words:

$$\mathbf{GapP} = \#\mathbf{P} - \#\mathbf{P} = \#\mathbf{P} - \mathbf{FP} = \mathbf{FP} = \#\mathbf{P}$$

- $(3) \Rightarrow \mathbf{GapP} \subseteq \mathbf{FP}^{\#\mathbf{P}[1]}$.

Characterizations of Complexity Classes

- **NP** consists of those languages L such that for some #**P** function f and all inputs x:
  - If $x \in L$ then $f(x) > 0$.
  - If $x \notin L$ then $f(x) = 0$.
- **UP** consists of those languages L such that for some #**P** function f and all inputs x:
  - If $x \in L$ then $f(x) = 1$.
  - If $x \notin L$ then $f(x) = 0$.
- **PP** consists of those languages L such that for some **GapP** function f and all inputs x:
  - If $x \in L$ then $f(x) > 0$.
  - If $x \notin L$ then $f(x) \leq 0$ (of $f(x) < 0$).
- **SPP** consists of those languages L such that for some **GapP** function f and all inputs x:
  - If $x \in L$ then $f(x) = 1$.
  - If $x \notin L$ then $f(x) = 0$.

Characterizations of Complexity Classes

- $\mathbf{C_{=}P}$ consists of those languages L such that for some **GapP** function f and all inputs x:
  - If $x \in L$ then $f(x) = 0$.
  - If $x \notin L$ then $f(x) \neq 0$ (or $f(x) > 0$).
- $\oplus\mathbf{P}$ consists of those languages L such that for some $\#\mathbf{P}$ function f and all inputs x:
  - If $x \in L$ then $f(x)$ is odd.
  - If $x \notin L$ then $f(x)$ is even.
- $\mathbf{Mod_k P}$ consists of those languages L such that for some $\#\mathbf{P}$ function f and all inputs x:
  - If $x \in L$ then $f(x) \mod k \neq 0$.
  - If $x \notin L$ then $f(x) \mod k = 0$.
- **MiddleP** consists of those languages L such that for some $\#\mathbf{P}$ function f and all inputs x:
  - If $x \in L$ then $\mathit{middle}(x) = 1$.
  - If $x \notin L$ then $\mathit{middle}(x) = 0$.

Characterizations of Complexity Classes

- **NP** consists of those languages L such that for some $\#\mathbf{P}$ function f and all inputs x:
  - If $x \in L$ then $f(x) > 0$.
  - If $x \notin L$ then $f(x) = 0$.

Similarly:

Class	Function f in:	If $x \in L$:	If $x \notin L$:
UP	$\#\mathbf{P}$	$f(x) = 1$	$f(x) = 0$
PP	**GapP**	$f(x) > 0$	$f(x) \leq 0$ or $f(x) < 0$
SPP	**GapP**	$f(x) = 1$	$f(x) = 0$
$\mathbf{C_{=}P}$	**GapP**	$f(x) = 0$	$f(x) \neq 0$ or $f(x) > 0$
$\oplus\mathbf{P}$	$\#\mathbf{P}$	$f(x)$ is odd	$f(x)$ is even
$\mathbf{Mod_k P}$	$\#\mathbf{P}$	$f(x) \mod k \neq 0$	$f(x) \mod k = 0$
MiddleP		$middle(x) = 1$	$middle(x) = 0$

Characterizations of Complexity Classes

- We define *middle* : $\{0,1\}^* \rightarrow \{0,1\}$ to return the $\lceil \frac{|x|}{2} \rceil$th bit of the string x.
- The class **MiddleP** consider the **middle bit** of a string, as **PP** consider the **high-order bit** and $\oplus$**P** the **low-order bit**.
- Observe that $\oplus\mathbf{P} = \mathbf{Mod_2P}$
- From the above we can easily have:
  - $\mathbf{NP} \subseteq co\mathbf{C_=P} \subseteq \mathbf{PP}$
  - $\mathbf{UP} \subseteq \mathbf{SPP}$
  - $\mathbf{C_=P} \subseteq \mathbf{PP}$
  - **PP** is closed under complement.

Characterizations of Complexity Classes

Theorem

$$\mathbf{P}^{\mathbf{PP}} = \mathbf{P}^{\mathbf{GapP}}$$

Proof:
We only need to show that every **GapP** function g is computable in $\mathbf{FP}^{\mathbf{PP}}$. If we consider the **GapP** function $f(x, k) = g(x) - k$, we have that $L \in \mathbf{PP}$ by the previous classification. One can use *binary search* using L as an oracle to find the value of $g(x)$. □

Counting vs Quantifiers

Theorem (Toda's Theorem)

$$\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}[1]}$$

- We can prove the following finer result:

$$\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{GapP}[1]} = \mathbf{P}^{\#\mathbf{P}[1]} = \mathbf{P}^{\mathbf{PP}[1]}$$

Toda's Theorem

Lemma

1. $\mathbf{PH} \subseteq \mathcal{P} \cdot \mathbf{PP}$
2. $\mathbf{PH} \subseteq \mathcal{P} \cdot \oplus\mathbf{P}$
3. $\mathbf{PH} \subseteq \mathcal{P} \cdot \mathbf{C_=P}$

Proof (2):

- Let $L \in \mathbf{PH}$. Then, $\chi_L(x) \in \mathbf{GapP^{PH}}$ and by using a Toda-Ogihara Theorem:
- $\exists\ g(x, r) \in \mathbf{GapP}$ and a polynomial q such that:

$$\mathbf{Pr}_{r \in \Sigma^q}\left[g(x, r) = L(x)\right] \geq \frac{3}{4}$$

Toda's Theorem

Proof (cont'd):

- If $x \in L$ then:

$$|\{r \in \Sigma^q \mid g(x,r) = 1\}| > |\{r \in \Sigma^q \mid g(x,r) \neq 1\}|$$

- If $x \notin L$ then:

$$|\{r \in \Sigma^q \mid g(x,r) = 0\}| > |\{r \in \Sigma^q \mid g(x,r) \neq 0\}|$$

- Let the set A consisting of $\langle x, r\rangle$ such that $g(x,r)$ is odd. Since $g(x,r)$ is in **GapP**, we have $A \in \oplus\mathbf{P}$.
- Let f the **GapP** function defined by $\#_A^p(x) - \#_{\overline{A}}^p(x)$.
- We have that L is in $\mathcal{P} \cdot \oplus\mathbf{P}$. $\square$

Toda's Theorem

Proof (of Toda's Theorem):

- Let $L \in \mathbf{PH}$.
- By the above lemma, we have that $L \in \mathcal{P} \cdot \oplus\mathbf{P}$
- In other words, $\exists g \in \mathbf{GapP}$ and a polynomial q such that $x \in L$ iff:

$$\underbrace{|\{r \in \Sigma^{q(n)} : g(x,r) mod 2 = 1\}|}_{R_1} > \underbrace{|\{r \in \Sigma^{q(n)} : g(x,r) mod 2 = 0\}|}_{R_0}$$

- Since this g does not lead directly to the proof of Toda's Theorem, we can use it to create a new function $\hat{g}$ in **GapP** with more useful properties:

Toda's Theorem

Lemma

For every polynomial p, there exists a **GapP** function $\hat{g}$ such that for all x and r:

1. If $g(x,r) mod 2 = 1$, then $\hat{g}(x,r) mod 2^{p(n)} = 1$
2. If $g(x,r) mod 2 = 0$, then $\hat{g}(x,r) mod 2^{p(n)} = 0$

Proof (of Toda's Theorem-cont'd):

- Let $p(n) = q(n) + 1$, and let $\hat{g}$ be the result of applying the above lemma to g.
- Then, consider the **GapP** function:

$$h(x) = \sum_{r \in \Sigma^{q(n)}} \hat{g}(x,r)$$

Toda's Theorem

Proof (of Toda's Theorem-cont'd):

- We have that $h(x) mod 2^{p(n)} = |R_1| mod 2^{p(n)} = |R_1|$, since $|R_1| \leq 2^{p(n)} < 2^{q(n)}$.
- We have that $x \in L$ if and only if $h(x) mod 2^{p(n)} > \frac{2^{q(n)}}{2}$.
- We can thus determine whether $x \in L$ by a single query to the **GapP** function h. □

Corollary

$$\mathbf{PH} \subseteq \mathcal{P} \cdot \oplus\mathbf{P} \subseteq \mathbf{MP}$$

Further Reading

- Sanjeev Arora and Boaz Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press, 1 edition, April 2009
- Oded Goldreich, *Computational Complexity: A Conceptual Perspective*, Cambridge University Press, 1 edition, April 2008
- Christos H. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994

- Lance Fortnow, *Counting Complexity*, In Complexity Theory Retrospective II, pages 81-107. Springer-Verlag, 1997
- Jacobo Torán,*Counting the number of solutions*, In Mathematical Foundations of Computer Science 1990, MFCS'90 Proceedings
- Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz, *Gap-definable counting classes*, J. Comput. Syst. Sci., 48(1):116-148, 1994

Thank You!